BITS Pilani
Pilani Campus

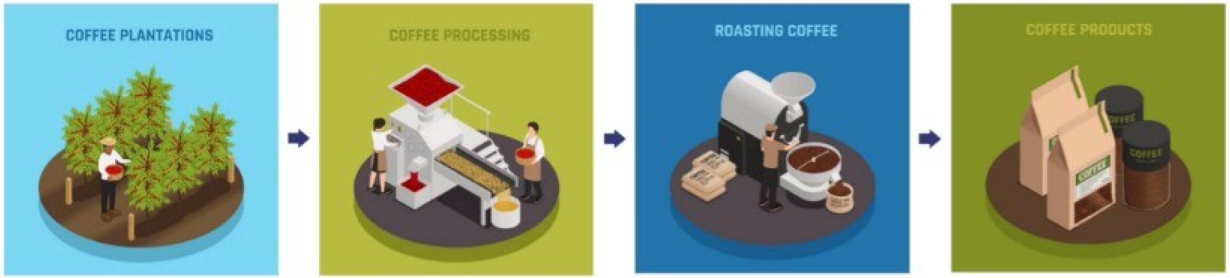
COMPUTING AND DESIGN
SESSION 1

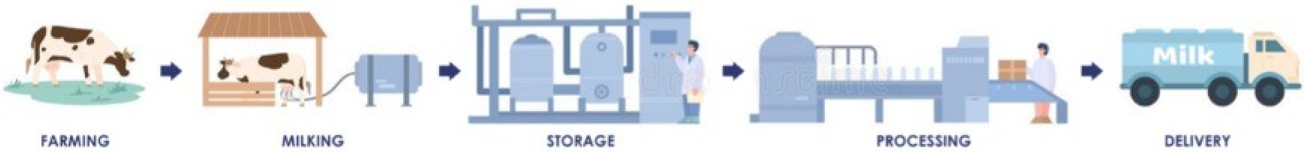
Faculty – Thangakumar J
WILP & Department of CS & IS

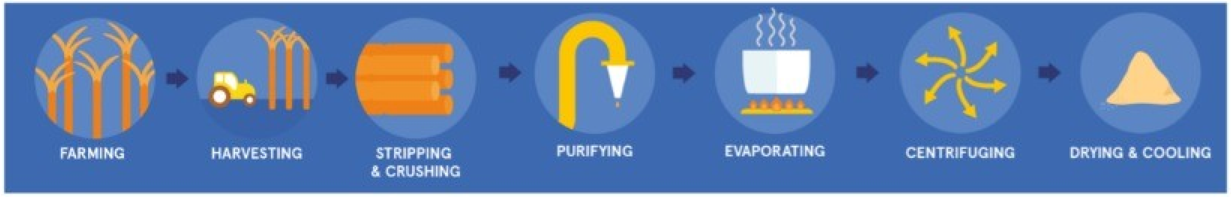
Today’s Session

List of Topic Title	Text/Ref Book/external resource
• **Introduction** • **Notion of Separation of Concerns** • **Systems and complexity** • **Why Modularity, Abstraction, Layering and Hierarchy aren’t enough**	R3 (Page 24-30) T2 (1.2, 1.3 and 1.5),

Introduction


COFFEE PLANTATIONS
COFFEE PROCESSING
ROASTING COFFEE
COFFEE PRODUCTS


FARMING
MILKING
STORAGE
PROCESSING
Milk
DELIVERY


FARMING
HARVESTING
STRIPPING & CRUSHING
PURIFYING
EVAPORATING
CENTRIFUGING
DRYING & COOLING

Coffee time !!!!

Why Product Development and Design needs Cohesion-Coupling?

- Ask any Engineer and they will explain in detail the importance of *Cohesion* and *Coupling*.
- But *Product Management* and *Design* have often sidelined these concepts as being too engineering-specific. However, in reality, many Product Principles are in fact centered around them.
- Thus unknowingly, we build products and make product decisions based on these very concepts.
- Therefore, let us dive into these core concepts and absorb them so that we can consciously make better product decisions and build better products.

Separation of concerns

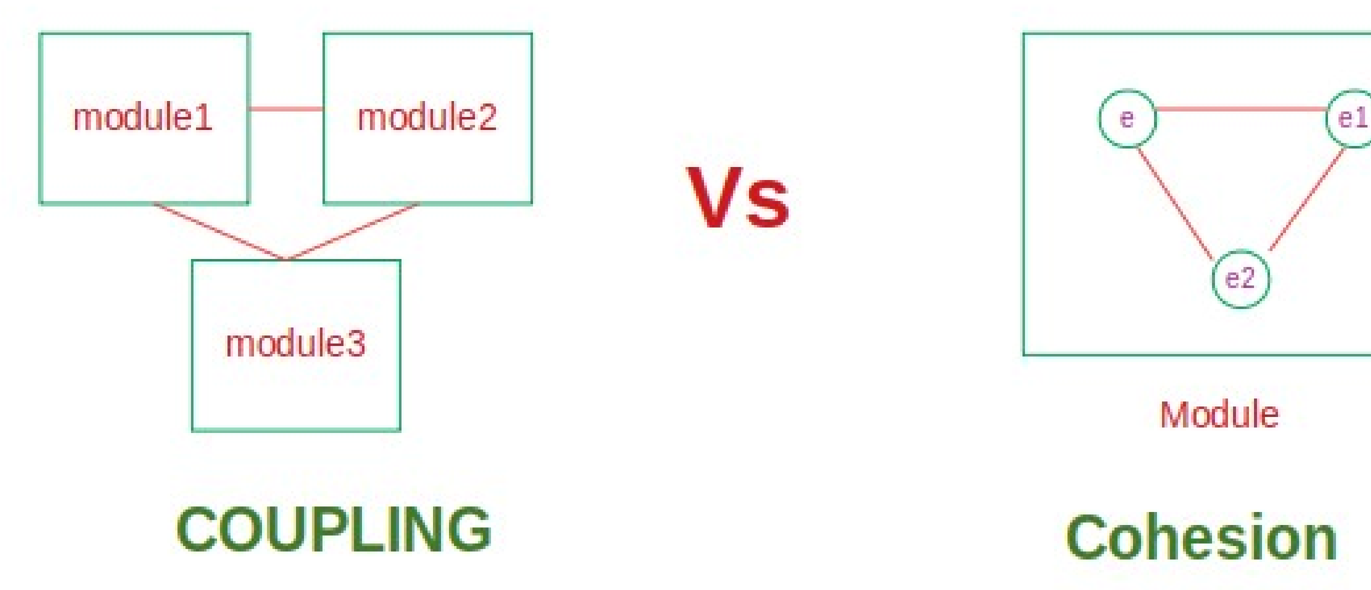
Software system must be decomposed into parts with minimal overlap in functionality. Methodology to follow when decomposing:

Coupling

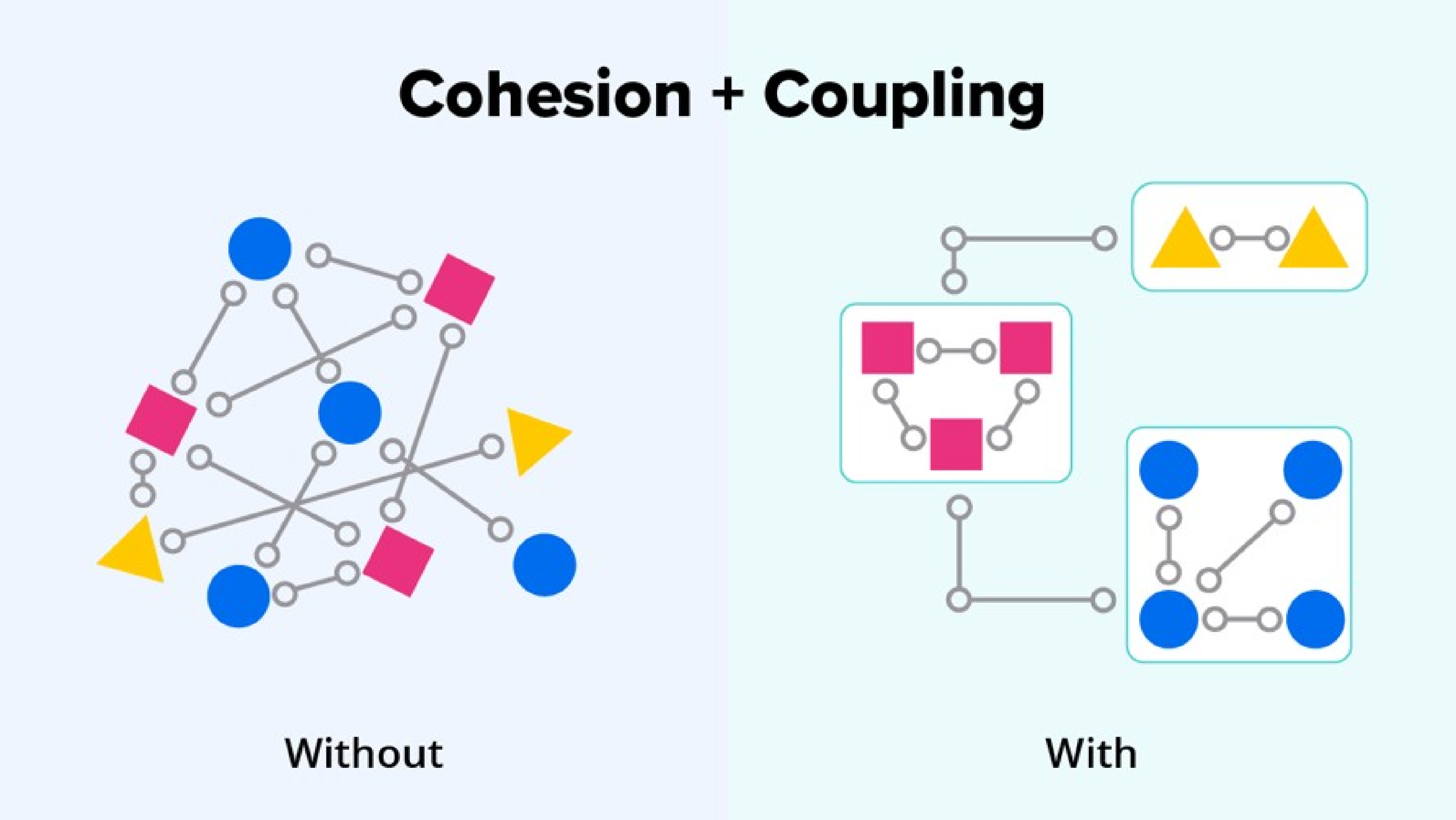
- The degree of dependency between two modules
- We always want low coupling

Cohesion

- The measure of how strongly-related is the set of functions performed by a module
- We always want high cohesion


module1
module2
module3
COUPLING
Vs
e
e1
e2
Module
Cohesion

Cohesion + Coupling


Cohesion + Coupling
Without
With

Any product can be broken down into two constituents:

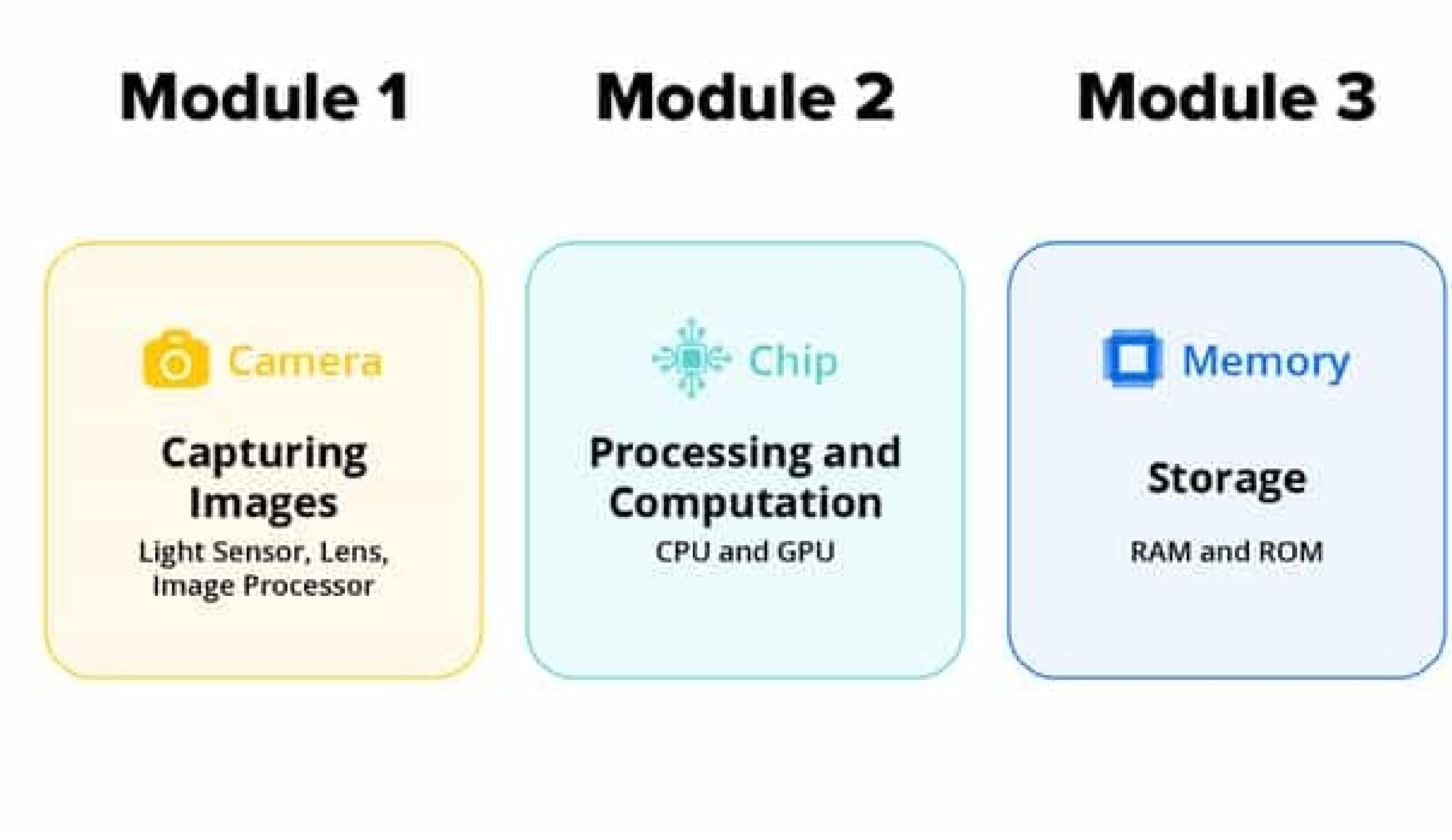
Components: *Individual elements that perform a single task.* In our *Smartphones*, individual components are RAM, ROM, Light Sensor, Lens, Image Processor, CPU, GPU, etc.

Modules: *Groups of functionally similar components.* The above *Smartphone* components are grouped together as Memory(RAM, ROM), Camera(Light Sensor, Lens, Image Processor), System-on-Chip(CPU, GPU), etc.

High Cohesion is when the components of a module are directed towards performing a single task. For example, components of the *Camera* module viz. Light Sensor, Lens, Image Processor are all directed towards performing a single task of capturing images. Similarly, the *Memory* and *Display* modules have *High Cohesion.*

Low cohesion is the exact opposite of High Cohesion. Here, each module ends up performing more than one task since its components are not directed towards a single task. Needless to say, we should always avoid *Low Cohesion*. But, more on that later.

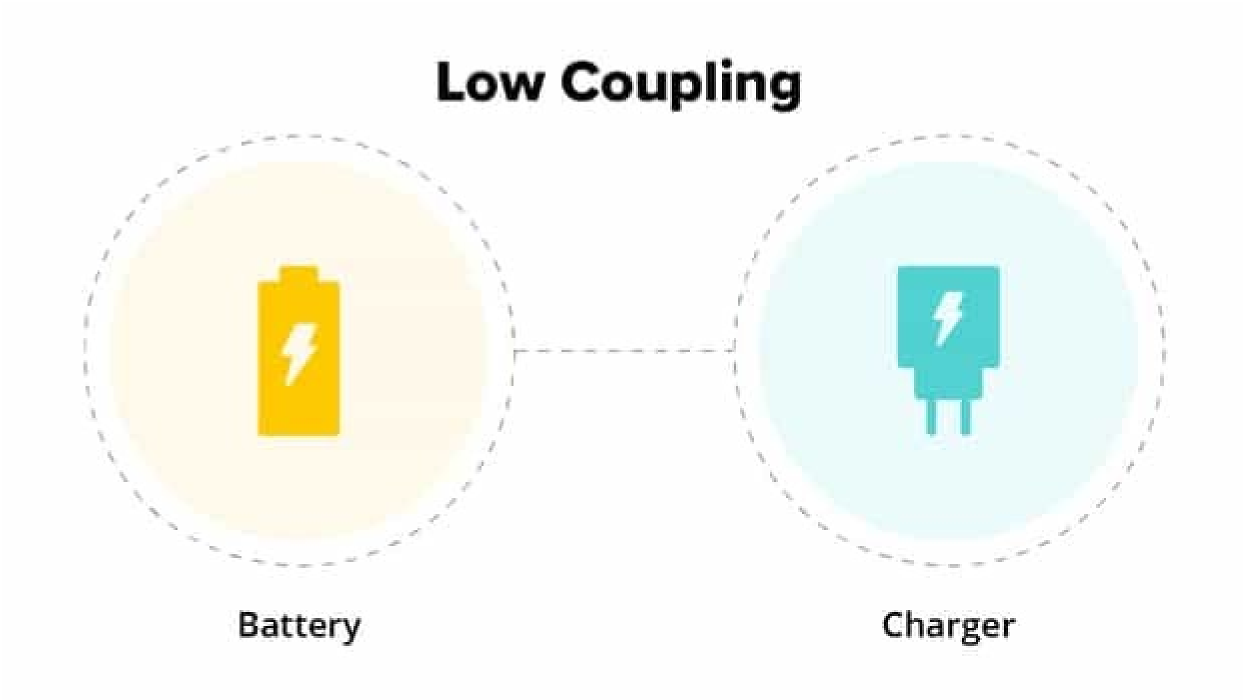
Cohesion:



Module 1	Module 2	Module 3
Camera	Chip	Memory
Capturing Images	**Processing and Computation**	**Storage**
Light Sensor, Lens, Image Processor	CPU and GPU	RAM and ROM

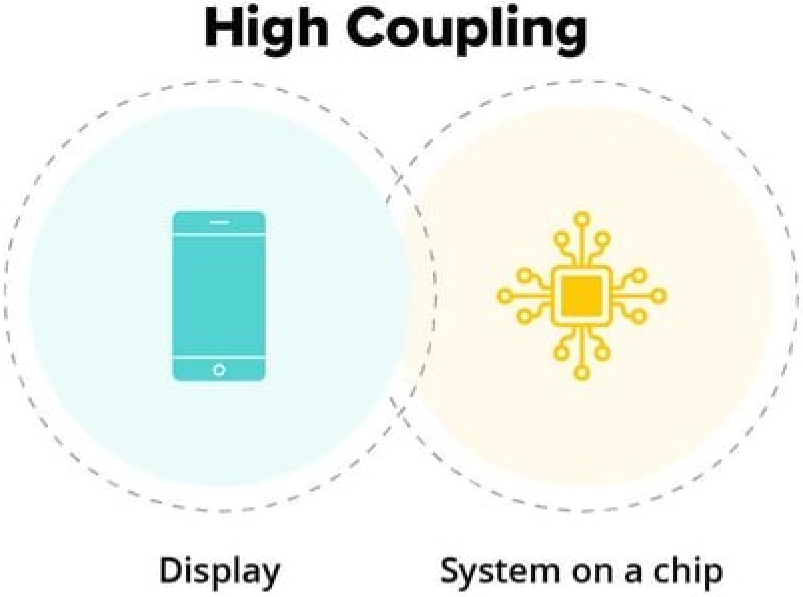
Coupling:

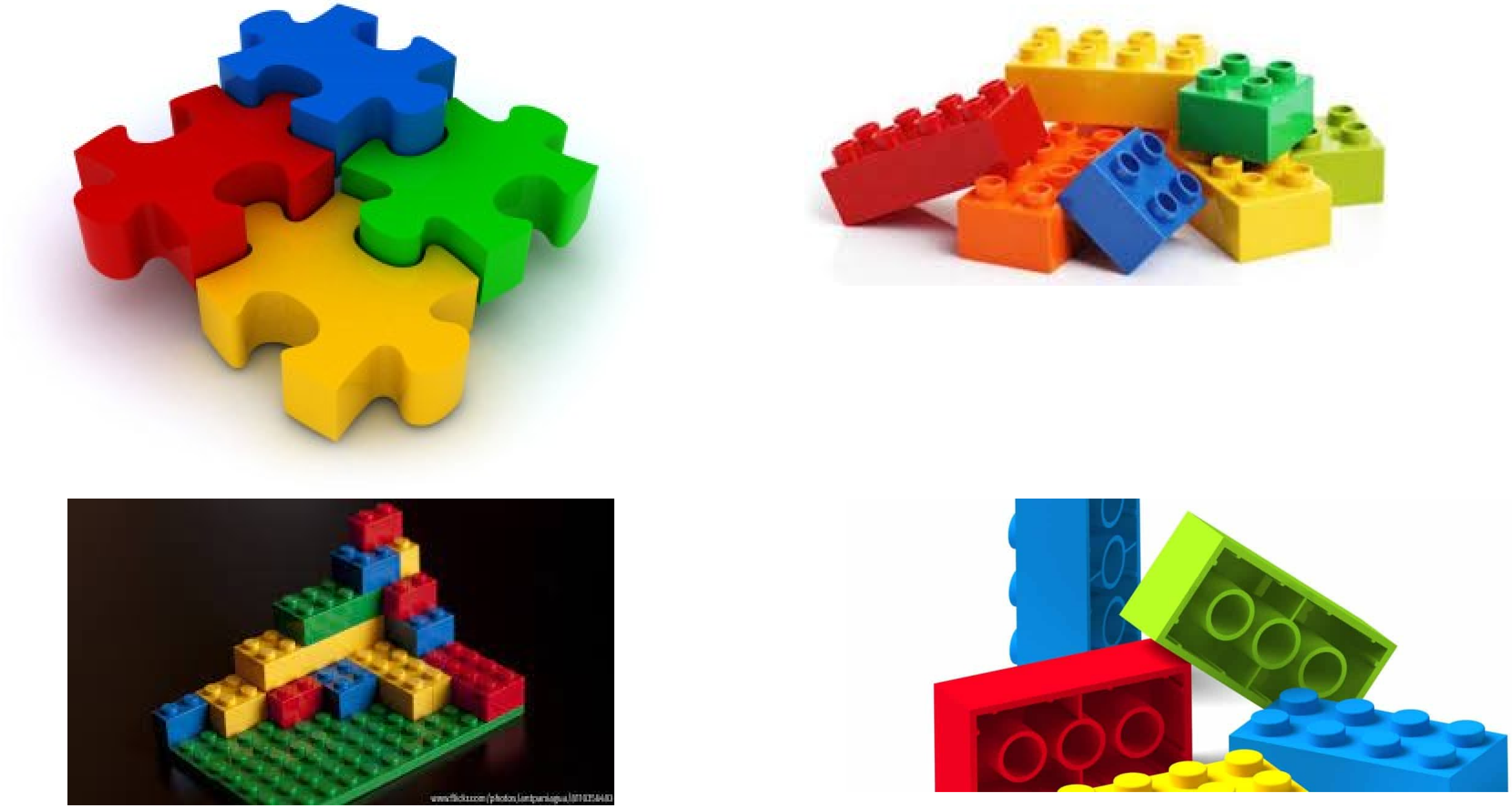
Low/Loose Coupling is when two modules are very lightly dependent on each other to perform their duties. For example, the *Battery* and *Battery-charger* are loosely coupled. If the *Battery-charger* gets corrupted, another *Battery-charger* can be used to charge the *Battery*, and if the *Battery* gets corrupted, then the same *Battery-charger* can still be used to charge other phones batteries.


Low Coupling
Battery
Charger

High/Tight Coupling is when two modules are heavily dependent on each other to perform their duties. For example, in contrast to the Battery and Battery charger, the *Display* module and *System-on-Chip* module are more tightly coupled.

If the *System-on-Chip* module gets corrupted, the *Display* module will not be able to perform its duties and if the *Display* gets corrupted, the *System-on-chip (GPU)* will not be able to perform its duties properly.


High Coupling
Display
System on a chip



Separation of concerns - Example of Classes using OOP

Student

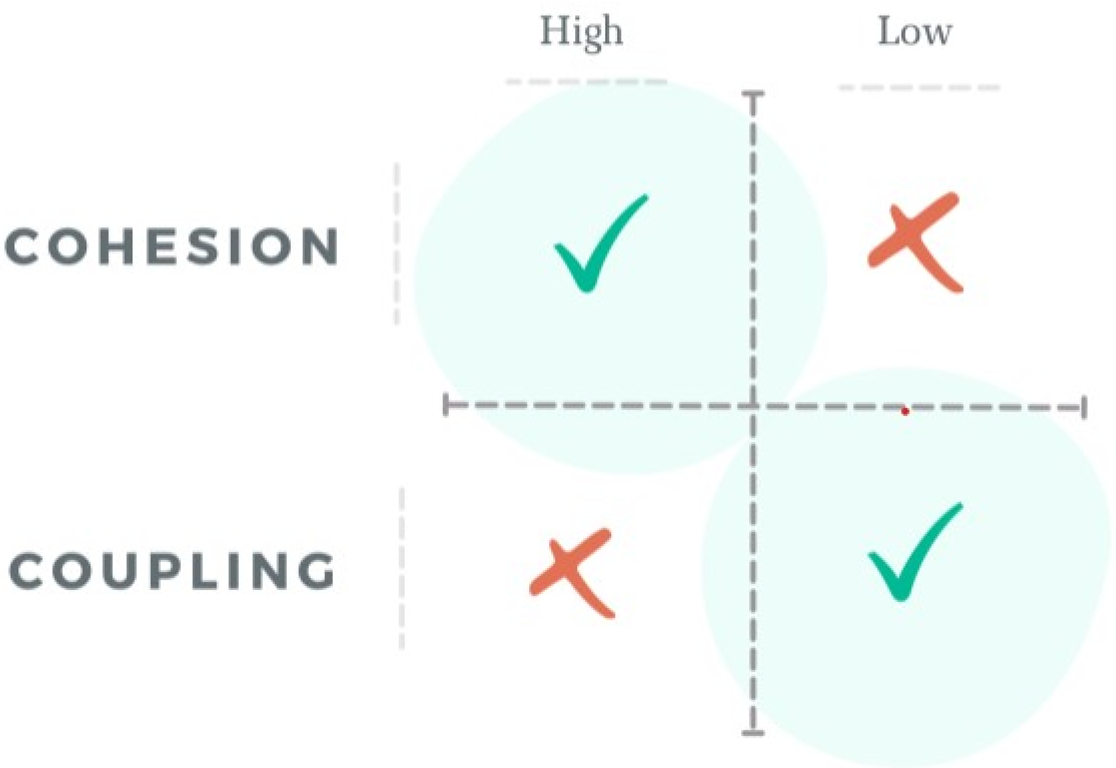
- ID
- Name
- Age
- Gender
- ….

Course

- ID
- Name
- Description
- Units
- …..

Cohesion :	Everything about Student or Course in respective classes
Coupling :	A change to student class should not impact course class

Better Maintenance ; Reusable; Extensible


High
Low
COHESION
COUPLING
✓
✗
✗
✓

In fact, it is a seesaw. If your modules have High Cohesion, *you naturally get* Low Coupling *between them. But, if you have* Low Cohesion, *your modules will have* High Coupling.

SYSTEMS AND COMPLEXITY

- A system is a set of interconnected components that has an expected behavior observed at the interface with its environment.
- Webster’s definition of “system” used the word “complex”.
- Looking up that term, we find that complex means “difficult to understand”.
- A set of signs of complexity that can help confirm a diagnosis here are five signs of complexity:
  - **Large number of components**
  - **Large number of interconnections**
  - **Many irregularities**
  - **A long description**
  - **A team of designers, implementers, or maintainers**

SYSTEMS AND COMPLEXITY (Contd.)

- The university library exhibits all five signs of complexity, but unanimity is not essential.
- On the other hand, the presence of only one or two of the signs may not make a compelling case for complexity.
- Systems considered in thermodynamics contain an unthinkably large number of components (elementary particles) and interactions, yet from the right point of view they do not qualify as complex because there is a simple, methodical description of their behavior.
- It is exactly when we lack such a simple, methodical description that we have complexity.

SYSTEMS AND COMPLEXITY (Contd.)

- One objection to conceiving complexity as being based on the five signs is that all systems are indefinitely, perhaps infinitely, complex because the deeper one digs the more signs of complexity turn up.
- Thus, even the simplest digital computer is made of gates, which are made with transistors, which are made of silicon, which is composed of protons, neutrons, and electrons, which are composed of quarks, which some physicists suggest are describable as vibrating strings, and so on.

And our coffee machine is not complex!

SOURCES OF COMPLEXITY - Cascading and Interacting Requirements

- There are many sources of complexity, but two merit special mention.
  - The first is in the number of requirements that the designer expects a system to meet.
  - The second is one particular requirement: maintaining high utilization.
- The underlying cause of this scenario is that the personal computer has been designed to meet many requirements: a well-organized file system, expandability of storage, ability to attach a variety of I/O devices, connection to a network, protection from malevolent persons elsewhere in the network, usability, reliability, low cost—the list goes on and on.
- Each of these requirements adds complexity of its own, and the interactions among them add still more complexity.

SOURCES OF COMPLEXITY - Cascading and Interacting Requirements

- The principle is subjective because complexity itself is subjective—its magnitude is in the mind of the beholder.
- Figure 1.1 provides a graphical interpretation of the principle.

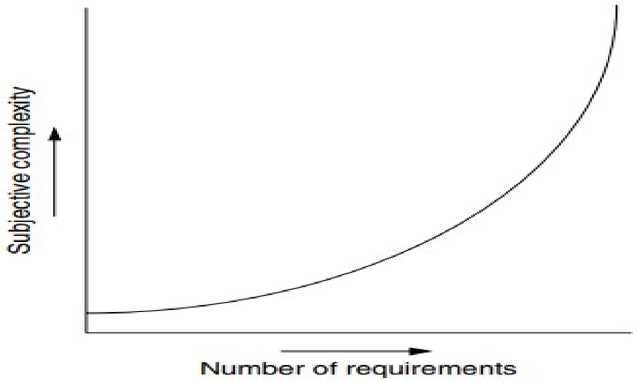


FIGURE 1.1

The principle of escalating complexity.

Maintaining High Utilization

- One requirement by itself is frequently a specific source of complexity.
- It starts with a desire for high performance or high efficiency.
- Whenever a scarce resource is involved, an effort arises to keep its utilization high.
- As a general rule, the more one tries to increase utilization of a limited resource, the greater the complexity (see Figure 1.2).
- The perceptive reader will notice that Figures 1.1 and 1.2 are identical.

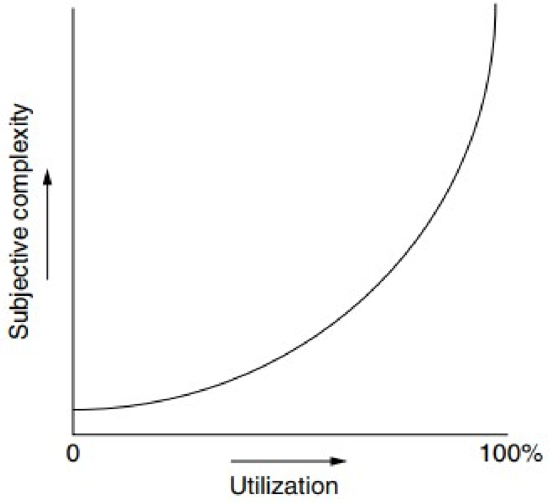


FIGURE 1.2

An example of diminishing returns: complexity grows with increasing utilization.

COPING WITH COMPLEXITY 1 - Modularity

- The simplest, most important tool for reducing complexity is the divide-and-conquer technique: analyze or design the system as a collection of interacting subsystems, called modules.
- The power of this technique lies primarily in being able to consider interactions among the components within a module without simultaneously thinking about the components that are inside other modules.
- To see the impact of reducing interactions,
  - consider the debugging of a large program with, say, N statements.
  - Assume that the number of bugs in the program is proportional to its size and the bugs are randomly distributed throughout the code.
  - The programmer compiles the program, runs it, notices a bug, finds and fixes the bug, and recompiles before looking for the next bug.
  - Assume also that the time it takes to find a bug in a program is roughly proportional to the size of the program.
- We can then model the time spent debugging: BugCount ~ N

DebugTime ~ N * BugCount

~ N^2

COPING WITH COMPLEXITY 1 - Modularity

- Unfortunately, the debugging time grows proportional to the square of the program size.
- Now suppose that the programmer divides the program into K modules, each of roughly equal size, so that each module contains N/K statements.
- To the extent that the modules implement independent features, one hopes that discovery of a bug usually will require examining only one module.
- The time required to debug any one module is thus reduced in two ways: the smaller module can be debugged faster, and since there are fewer bugs in smaller programs, any one module will not need to be debugged as many times.
- These two effects are partially offset by the need to debug all K modules.
- Thus our model of the time required to debug the system of K modules becomes

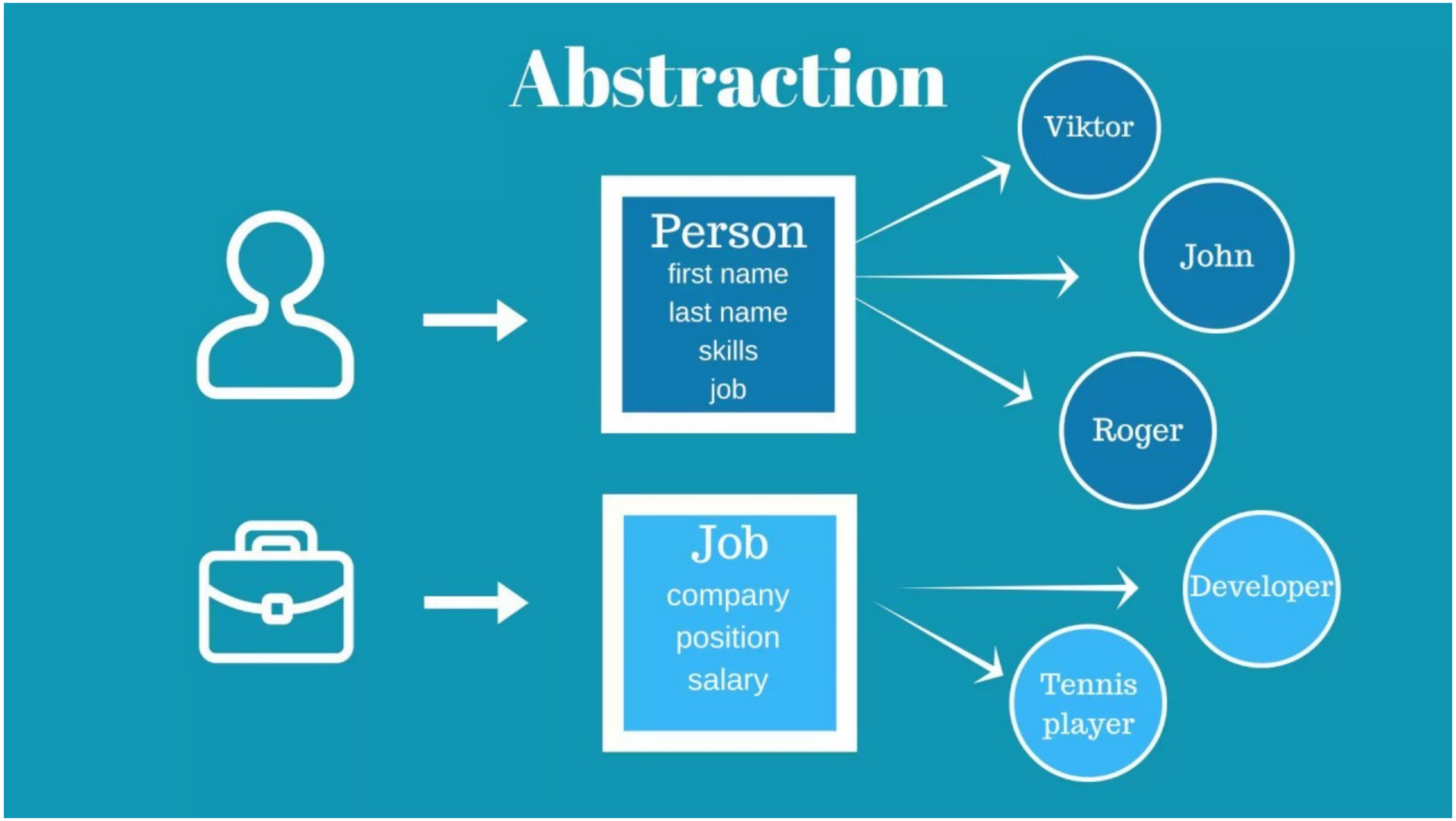
$$\text{DebugTime} \sim (N/K)^2 * K$$

$$\sim N^2/K$$

COPING WITH COMPLEXITY 1 - Modularity

- Modularization into K components thus reduces debugging time by a factor of K.
- Although the detailed mechanism by which modularity reduces effort differs from system to system, this property of modularity is universal.
- For this reason, one finds modularity in every large system.
- The feature of modularity that we are taking advantage of here is that it is easy to replace an inferior module with an improved one, thus allowing incremental improvement of a system without completely rebuilding it.
- Modularity thus helps control the complexity caused by change.
- This feature applies not only to debugging but to all aspects of system improvement and evolution.
- The reason is that once an interface has been used by another module, changing the interface requires replacing at least two modules.
- If an interface is used by many modules, changing it requires replacing all of those modules simultaneously. For this reason, it is particularly important to get the modularity right.

Abstraction

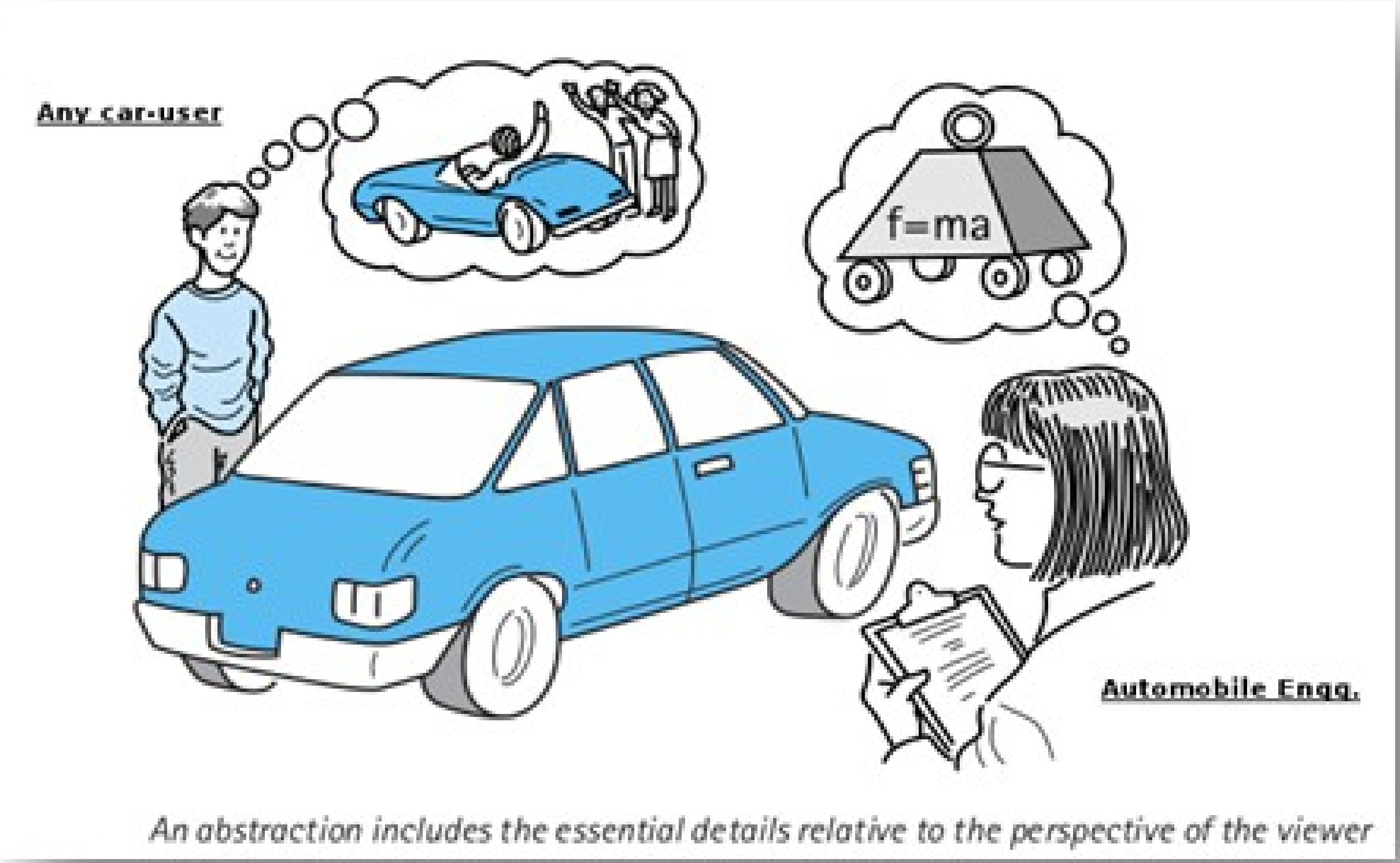

Abstraction
Person
first name
last name
skills
job
Viktor
John
Roger
Job
company
position
salary
Developer
Tennis player

What does the above abstract picture notify?

Abstraction can be found in practically all real-world machines.

- Your automobile is an excellent illustration of abstraction. Turning the key or pressing the start button is how you start a car. You don't need to know how your car's engine starts or what components it contains. The user is completely unaware of the car's internal implementation and complicated logic.

- We can heat our food in Microwave. We press some buttons to set the timer and type of food. Finally, we get a hot and delicious meal. The microwave internal details are hidden from us. We have been given access to the functionality in a very simple manner.

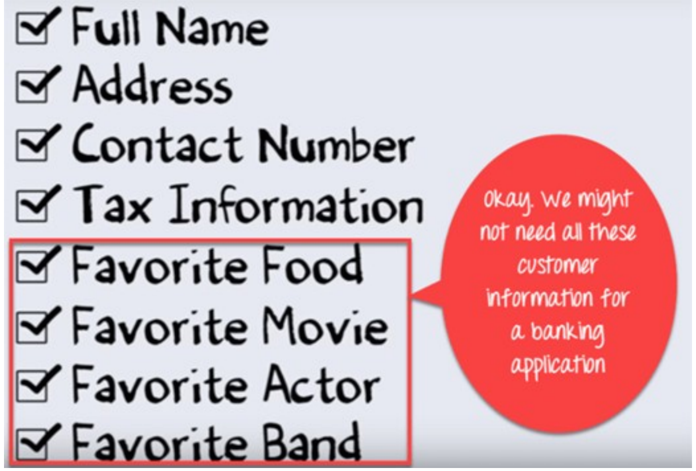

Any car-user
f=ma
Automobile Engg.

An abstraction includes the essential details relative to the perspective of the viewer

Types of Abstraction

- There are two types of abstraction.
- Data Abstraction
- Process Abstraction
- Data Abstraction

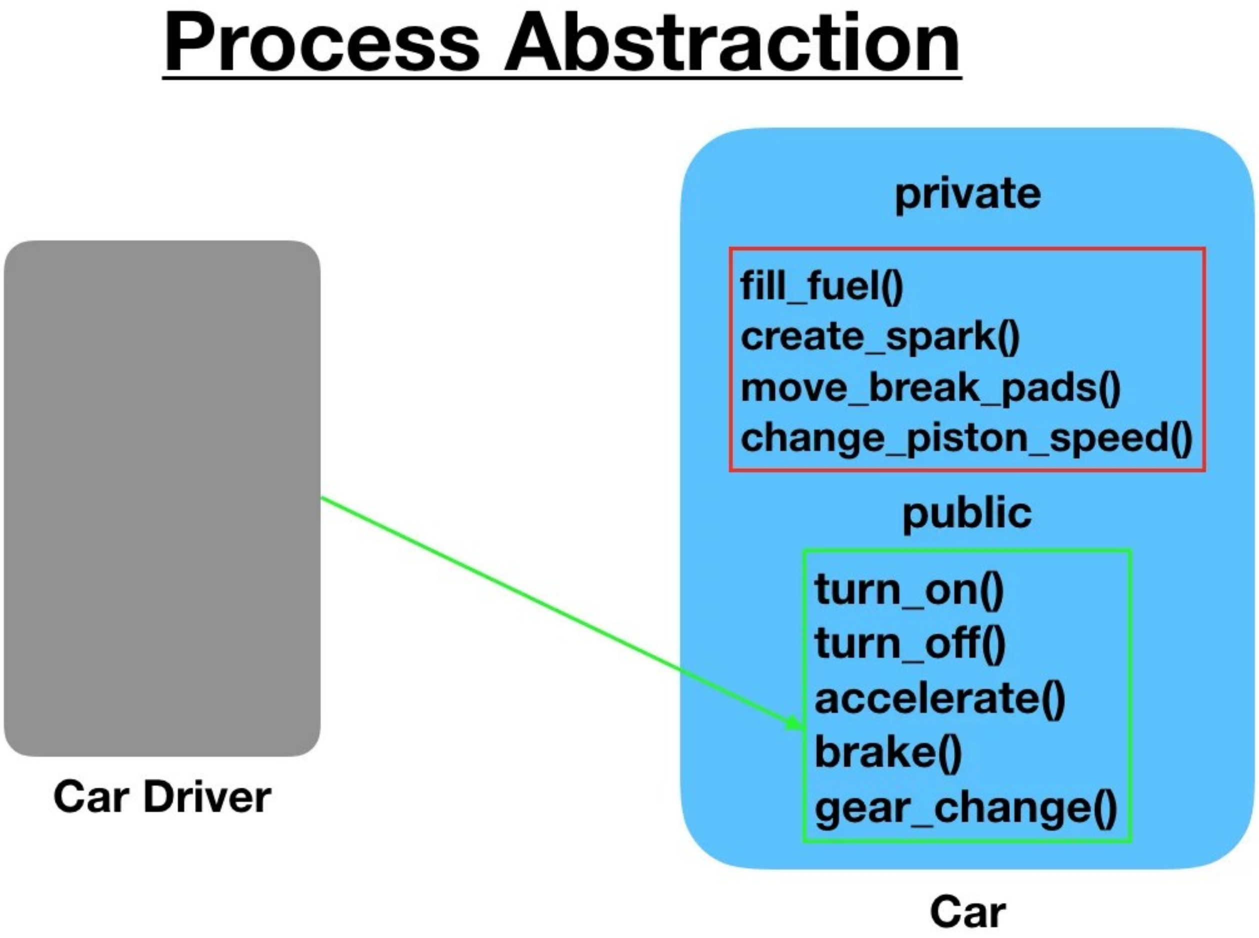
When the object data is not visible to the outer world, it creates data abstraction. If needed, access to the Objects' data is provided through some method



☑ Full Name
☑ Address
☑ Contact Number
☑ Tax Information
☑ Favorite Food
☑ Favorite Movie
☑ Favorite Actor
☑ Favorite Band

Okay, we might not need all these customer information for a banking application

Process abstraction



Process Abstraction

private

- fill_fuel()
- create_spark()
- move_break_pads()
- change_piston_speed()

public

- turn_on()
- turn_off()
- accelerate()
- brake()
- gear_change()

Car Driver

Car

Process Abstraction

- A piece of software is essentially a set of statements written in any programming language. The majority of the time, statements are similar and are repeated in several locations.

- The process of finding all such assertions and exposing them as a unit of work is known as process abstraction. When we build a function to execute any task, we usually use this feature.

Abstraction

- An important assumption in the numerical example of the effect of modularity on debugging time may not hold up in practice: that discovery of a bug should usually lead to examining just one module.
- Abstraction is separation of interface from internals, of specification from implementation.
- Because abstraction nearly always accompanies modularity, some authors do not make any distinction between the two ideas.
- One sometimes sees the term functional modularity used to mean modularity with abstraction.
- The general ability of sequential circuits to remember state is abstracted into particular, easy-to-describe modules called registers.
- Programs are designed to hide details of their representation of complex data structures and details of which other programs they call.

COPING WITH COMPLEXITY 1 - Modularity

- The goal of minimizing interconnections among modules may be defeated if unintentional or accidental interconnections occur as a result of implementation errors
- Software is particularly subject to this problem because the modular boundaries provided by separately compiled subprograms are somewhat soft and easily penetrated by errors in using pointers, filling buffers, or calculating array indices.
- For this reason, system designers prefer techniques that enforce modularity by interposing impenetrable walls between modules.
- The robustness principle is one of the key ideas underlying modern mass production.
- The breakthrough came with the realization that if one specified tolerances for components and designed each component to mate with any other component that was within its specified tolerance, then it would be possible to modularize and speed up manufacturing by having interchangeable parts.

COPING WITH COMPLEXITY 1 - Modularity

- Some systems implement the safety margin principle by providing two modes of operation, which might be called “shake-out” and “production”.
  - In shake-out mode, modules check every input carefully and refuse to accept anything that is even slightly out of specification, thus allowing immediate discovery of problems and of programming errors near their source.
  - In production mode, modules accept any input that they can reasonably interpret, in accordance with the robustness principle.
- Carefully designed systems blend the two ideas: accept any reasonable input but report any input that is beginning to drift out of tolerance so that it may be repaired before it becomes completely unusable.

Layering

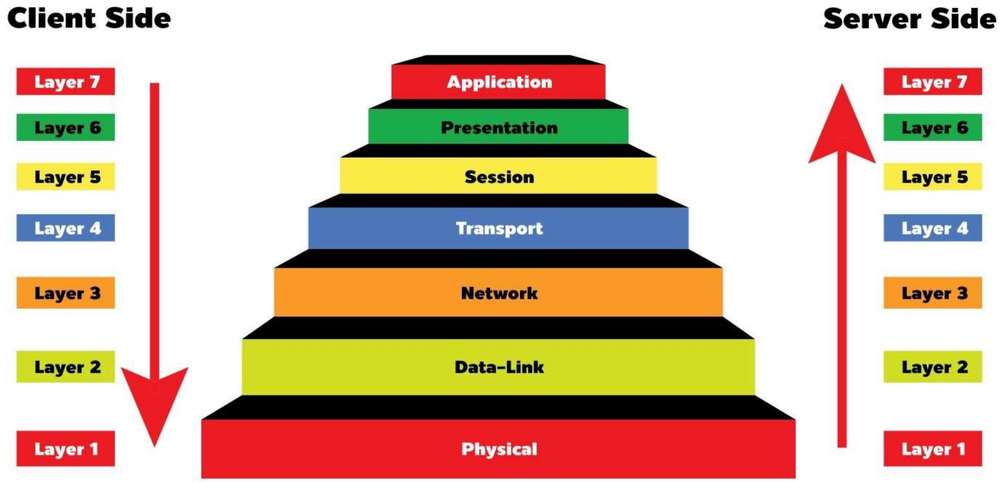
- One powerful way to reduce module interconnections is to employ a particular method of module organization known as layering.
- In designing with layers, one builds on a set of mechanisms that is already complete (a lower layer) and uses them to create a different complete set of mechanisms (an upper layer).
- A layer may itself be implemented as several modules, but as a general rule, a module of a given layer interacts only with its peers in the same layer and with the modules of the next higher and next lower layers.
- That restriction can significantly reduce the number of potential inter-module interactions in a big system.
- Some of the best examples of this approach are found in computer systems: an interpreter for a high-level language is implemented using a lower-level, more machine-oriented, language.

COPING WITH COMPLEXITY 1 - Modularity

- Thus, nearly every computer system comprises several layers.
- The lowest layer consists of gates and memory cells, upon which is built a layer consisting of a processor and memory.
- On top of this layer is built an operating system layer, which acts as an augmentation of the processor and memory layer.
- Finally, an application program executes on this augmented processor and memory layer.
- In each layer, the functions provided by the layer below are rearranged, repackaged, re-abstracted, and reinterpreted as appropriate for the convenience of the layer above.

Level 6	User	Executable Programs
Level 5	High Level Language	C++ , Java
Level 4	Assembly Language	Assembly Code
Level 3	System Software	Operating System
Level 2	Machine	Instruction Set Architecture
Level 1	Control	Microcode
Level 0	Digital Logic	Circuits , Gates

OSI MODEL


Client Side
Server Side
Layer 7
Layer 6
Layer 5
Layer 4
Layer 3
Layer 2
Layer 1
Application
Presentation
Session
Transport
Network
Data-Link
Physical
Layer 7
Layer 6
Layer 5
Layer 4
Layer 3
Layer 2
Layer 1

- Computer System Level Hierarchy is the combination of different levels that connects the computer with the user and that makes the use of the computer. It also describes how the computational activities are performed on the computer and it shows all the elements used in different levels of system.

- **Level-0:**
  It is related to digital logic. Digital logic is the basis for digital computing and provides a fundamental understanding of how circuits and hardware communicate within a computer. It consists of various circuits and gates etc.
- **Level-1:**
  This level is related to control. Control is the level where microcode is used in the system. Control units are included in this level of the computer system.

- **Level-2:**
  This level consists of machines. Different types of hardware are used in the computer system to perform different types of activities. It contains instruction set architecture.
- **Level-3:**
  System software is a part of this level. System software is of various types. System software mainly helps in operating the process and it establishes the connection between hardware and user interface. It may consist operating system, library code, etc.

- **Level-4:**
  Assembly language is the next level of the computer system. The machine understands only the assembly language and hence in order, all the high-level languages are changed in the assembly language. Assembly code is written for it.
- **Level-5:**
  This level of the system contains high-level language. High-level language consists of C++, Java, FORTRAN, and many other languages. This is the language in which the user gives the command.
- Level-6:

This is the last level of the computer system hierarchy. This consists of users and executable programs.

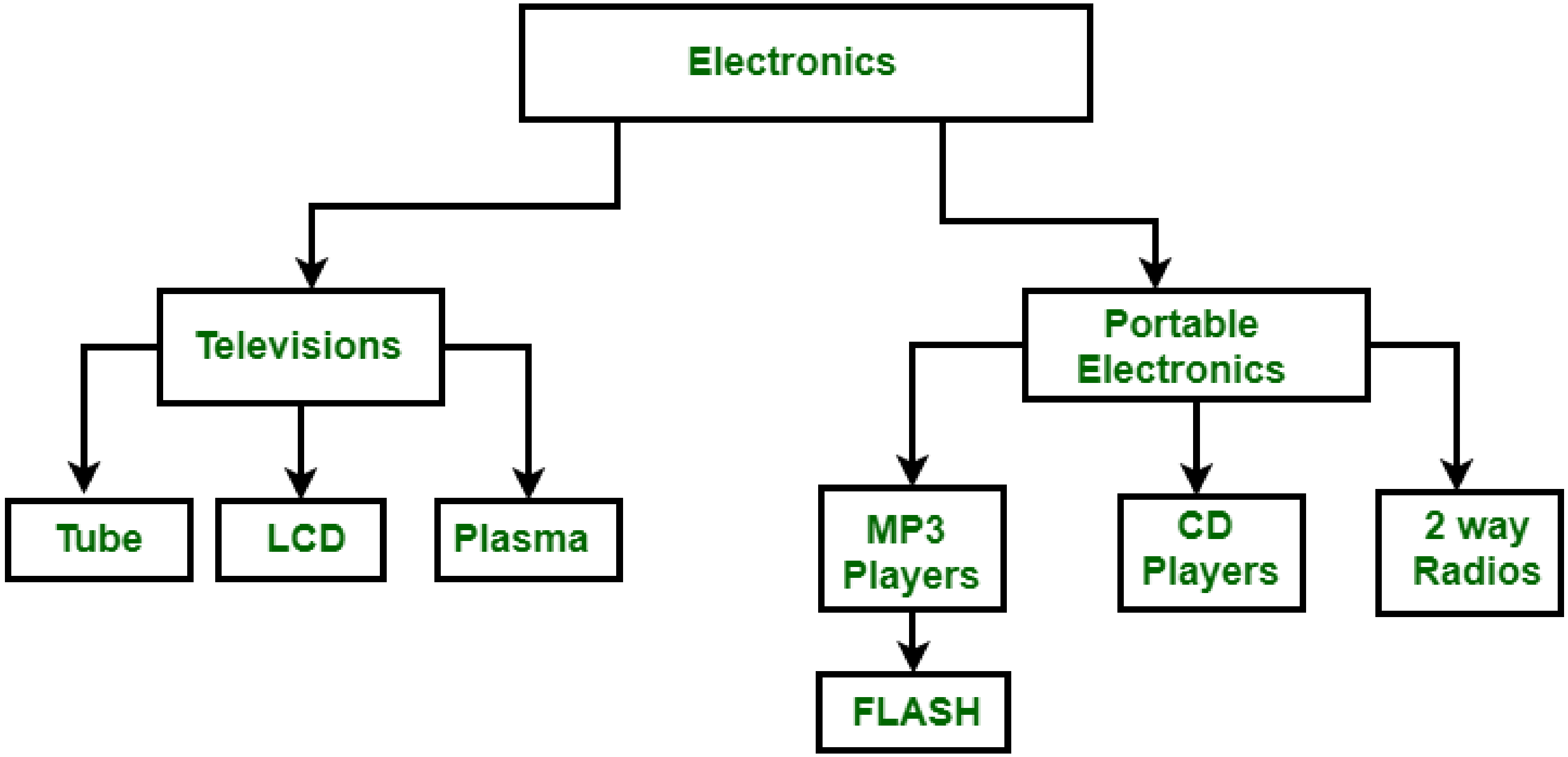
Hierarchy

- The final major technique for coping with complexity also reduces interconnections among modules but in a different, specialized way.
- Start with a small group of modules, and assemble them into a stable, self-contained subsystem that has a well-defined interface.
- Next, assemble a small group of subsystems to produce a larger subsystem.
- This process continues until the final system has been constructed from a small number of relatively large subsystems.
- The result is a tree-like structure known as a hierarchy.
- Hierarchy constrains a system of N components, which in the worst case might exhibit N * (N - 1) interactions, so that each component can interact only with members of its own subsystem, except for an interface component that also interacts with other members of the subsystem at the next higher level of hierarchy.

COPING WITH COMPLEXITY 1 - Hierarchy

- Analogous to the way that modularity reduces the effort of debugging, hierarchy reduces the number of potential interactions among modules from square-law to linear.
- This effect is most strongly noticed by the designer of an individual module.
- If there are no constraints, each module should in principle be prepared to interact with every other module of the system.
- The advantage of a hierarchy is that the module designer can focus just on interactions with the interfaces of other members of its immediate subsystem.

Hierarchy


Electronics

Televisions

Portable Electronics

Tube

LCD

Plasma

MP3 Players

CD Players

2 way Radios

FLASH

COPING WITH COMPLEXITY II

- Modest physical limits in hardware and very distant physical limits in software together give us the opportunity to create systems of unimaginable—and unmanageable— complexity, and the rapid pace of technology change tempts designers to deliver systems using new and untested ground rules.
- These two effects amplify the complexity of computer systems when compared with systems from other engineering areas.
- Thus, computer system designers need some additional tools to cope with complexity.

Why modularity, abstraction, layering, and hierarchy aren’t enough

- Modularity, abstraction, layering, and hierarchy are a major help, but by themselves they aren’t enough to keep the resulting complexity under control.
  - The right modularity from a sea of plausible alternative modularity
  - The right abstraction from a sea of plausible alternative abstractions
  - The right layering from a sea of plausible alternative layering
  - The right hierarchy from a sea of plausible alternative hierarchies
- Although some design principles are available, they are far too few, and the only real guidance comes from experience with previous systems.

Iteration

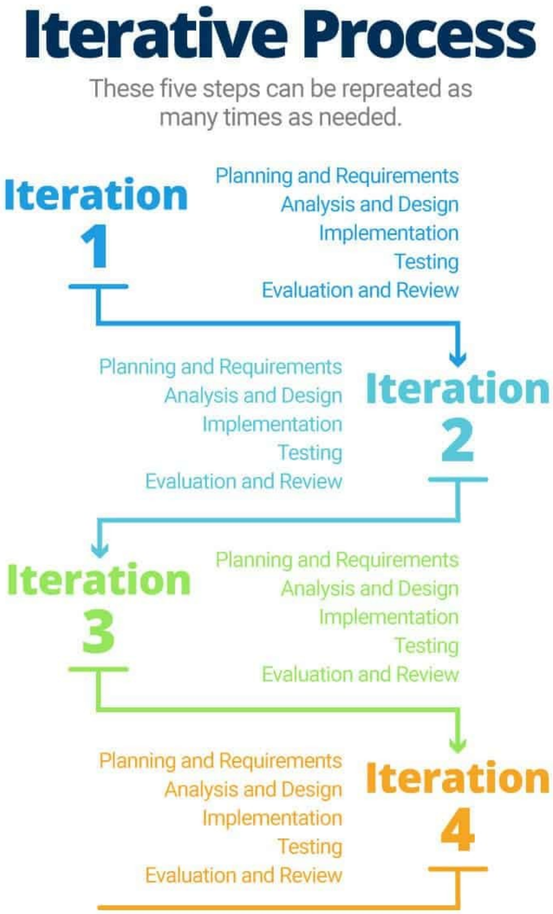
- The essence of iteration is to start by building a simple, working system that meets only a modest subset of the requirements and then evolve that system in small steps to gradually encompass more and more of the full set of requirements.
- The idea is that small steps can help reduce the risk that complexity will overwhelm a system design.
- Having a working system available at all times helps provide assurance that something can be built and provides on-going experience with the current technology ground rules as well as an opportunity to discover and fix bugs.
- Finally, adjustments for technology changes that arrive during the system development are easier to incorporate as part of one or more of the iterations.

Iteration

- Document the assumptions behind the design so that when the time comes to change the design you can more easily figure out what else has to change. Expect not only to modify and replace modules, but also to re-modularize as the system and its requirements become better understood.
  - Take small steps
  - Don’t rush
  - Plan for feedback
  - Study failures
- Iteration sounds like a straightforward technique, but several obstacles tend to interfere with it. The main obstacle is that as a design evolves through a series of iterations, a risk of losing conceptual integrity arises.
- In most organizations, good news flows rapidly throughout the organization, but bad news often gets confined to the part of the organization that discovers it, at least until it can fix the problem and report good news.

Iteration

- This phenomenon, the bad-news diode, can prevent realization that changing a different part of the system is more appropriate.
- A longer-term risk of iteration sometimes shows up when the initial design is both simple and successful. Success can lead designers to be overconfident and to be too ambitious on a later iteration.
- Technology has improved in the time since deployment of the initial version of the system and feedback has suggested lots of new features.
- Each suggested feature looks straightforward by itself, and it is difficult to judge how they might interact.
- The result is often a disastrous overreaching and consequent failure that is so common that it has a name: the second-system effect.
- Iteration can be thought of as applying modularity to the management of the system design and implementation process.



Iterative Process

These five steps can be repreated as many times as needed.

Iteration 1

- Planning and Requirements
- Analysis and Design
- Implementation
- Testing
- Evaluation and Review

Iteration 2

- Planning and Requirements
- Analysis and Design
- Implementation
- Testing
- Evaluation and Review

Iteration 3

- Planning and Requirements
- Analysis and Design
- Implementation
- Testing
- Evaluation and Review

Iteration 4

- Planning and Requirements
- Analysis and Design
- Implementation
- Testing
- Evaluation and Review

- **Step One: Planning and Requirements:** In this stage, map out the initial requirements, gather the related documents, and create a plan and timeline for the first iterative cycle.
- **Step Two: Analysis and Design:** Finalize the business needs, database models, and technical requirements based on the plan. Create a working architecture, schematic, or algorithm that satisfies your requirements.
- **Step Three: Implementation:** Develop the functionality and design required to meet the specifications.
- **Step Four: Testing:** Identify and locate what’s not working or performing to expectations. Stakeholders, users, and product testers weigh in with their experience.
- **Step Five: Evaluation and Review:** Compare this iteration with the requirements and expectations.

- After you complete these steps, it’s time to tackle the next cycle. In the iterative process, the product goes back to step one to build on what’s working. Identify what you learned from the previous iteration.
- This iterative development, sometimes called *circular* or *evolutionary development*, is based on refining the first version through subsequent cycles, especially as you gather and include requirements. It allows you to remain flexible as you identify new needs or unexpected business issues.

Keep it simple

- Remarkably, one of the most effective techniques in coping with complexity is also one that is most difficult to apply: simplicity.
  - Previous systems give a taste of how great things could be if more features were added.
  - the technology has improved so much that cost and performance are not constraints.
  - each of the suggested new features has been successfully demonstrated somewhere.
  - none of the exceptions or other complications seems by itself to be especially hard to deal with.
  - there is fear that a competitor will market a system that has even more features.
  - among system designers, arrogance, pride, and overconfidence are more common than clear awareness of the dangers of complexity.
-

THANK YOU

BITS Pilani
Pilani Campus

BITS Pilani
Pilani Campus

COMPUTING AND DESIGN

SESSION 2

Course design by - Dr. Lucy J. Gudino &
Dr Chandrasekhar
Faculty – Thangakumar J
WILP & Department of CS & IS

Last Session

List of Topic Title	Text/Ref Book/external resource
• **Introduction** • **Notion of Separation of Concerns** • **Systems and complexity** • **Why Modularity, Abstraction, Layering and Hierarchy aren’t enough**	R3 (Page 24-30) T2 (1.2, 1.3 and 1.5),

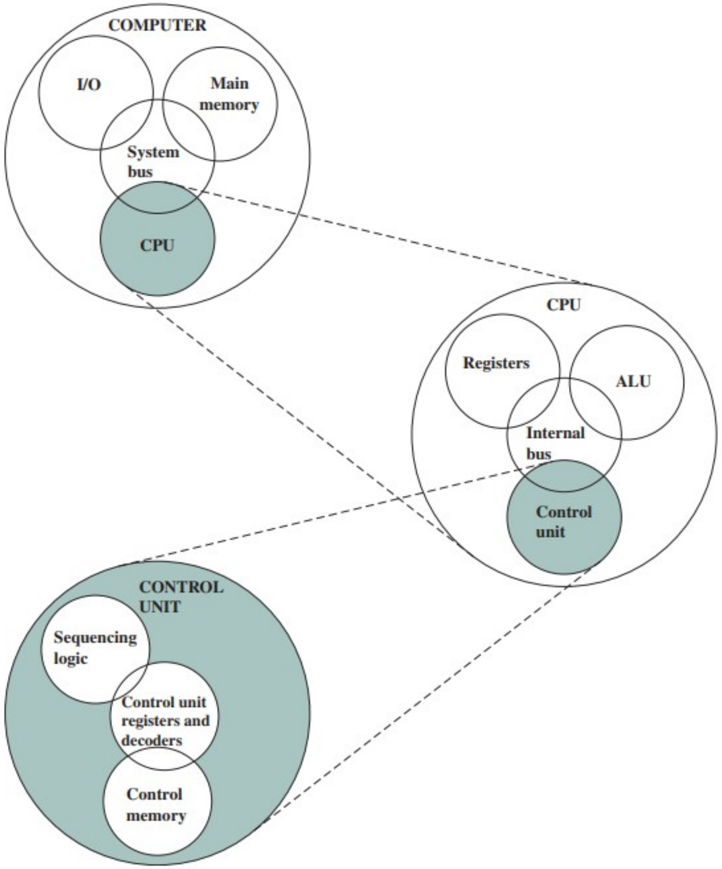
ABSTRACTION & THE NOTION OF INTERFACE

Today's topic

- **Major Modules of a Typical Computer System: Processor(s), Memory, I/O and Storage Devices**
- **Hardware abstraction vs. Software**
- **The three fundamental Abstractions**
  - Memory
  - Interpreters
  - Communication links
- **Organizing Computer Systems**
  - A Hardware Layer : The Bus
  - A Software Layer : The File Abstraction

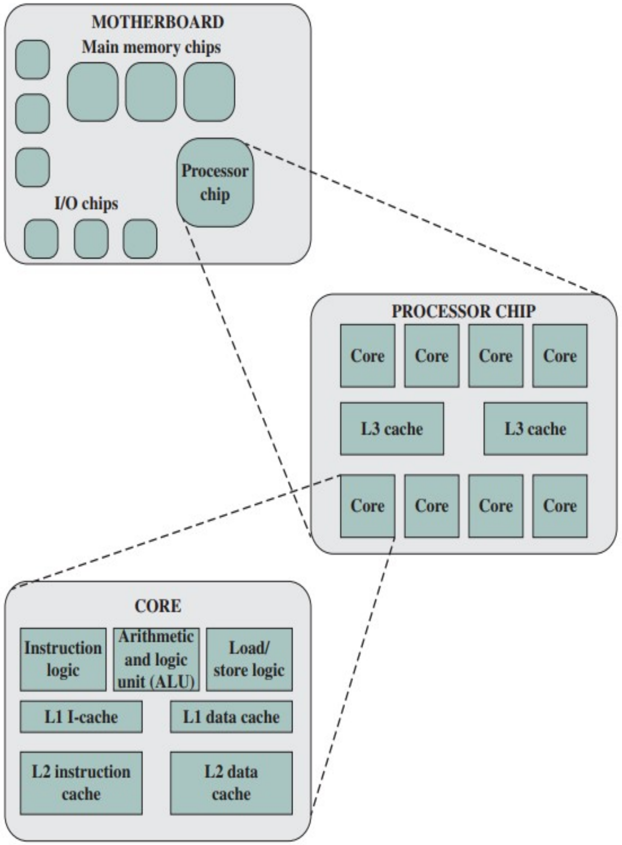
Computer: Top-level Structure

- Figure provides a hierarchical view of the internal structure of a traditional single-processor computer.
- The main structural components:
  - **Central processing unit (CPU)**
  - **Main memory**
  - **System interconnection**


COMPUTER
I/O
Main memory
System bus
CPU
CPU
Registers
ALU
Internal bus
Control unit
CONTROL UNIT
Sequencing logic
Control unit registers and decoders
Control memory

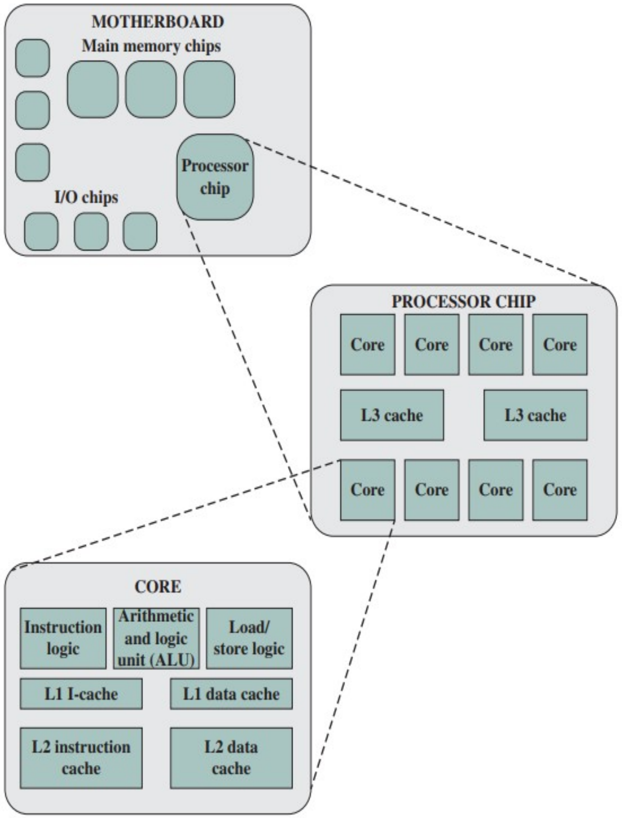
Simplified View of Major Elements of a Multicore Computer

- Figure is a simplified view of the principal components of a typical multicore computer.
- A printed circuit board (PCB) is a rigid, flat board that holds and interconnects chips and other electronic components.
- The board is made of layers, typically two to ten that interconnect components via copper pathways that are etched into the board.


MOTHERBOARD
Main memory chips
Processor chip
I/O chips
PROCESSOR CHIP
Core
Core
Core
Core
L3 cache
L3 cache
Core
Core
Core
Core
CORE
Instruction logic
Arithmetic and logic unit (ALU)
Load/ store logic
L1 I-cache
L1 data cache
L2 instruction cache
L2 data cache

Simplified View of Major Elements of a Multicore Computer

- The main printed circuit board in a computer is called a system board or motherboard, while smaller ones that plug into the slots in the main board are called expansion boards.
- The most prominent elements on the motherboard are the chips.

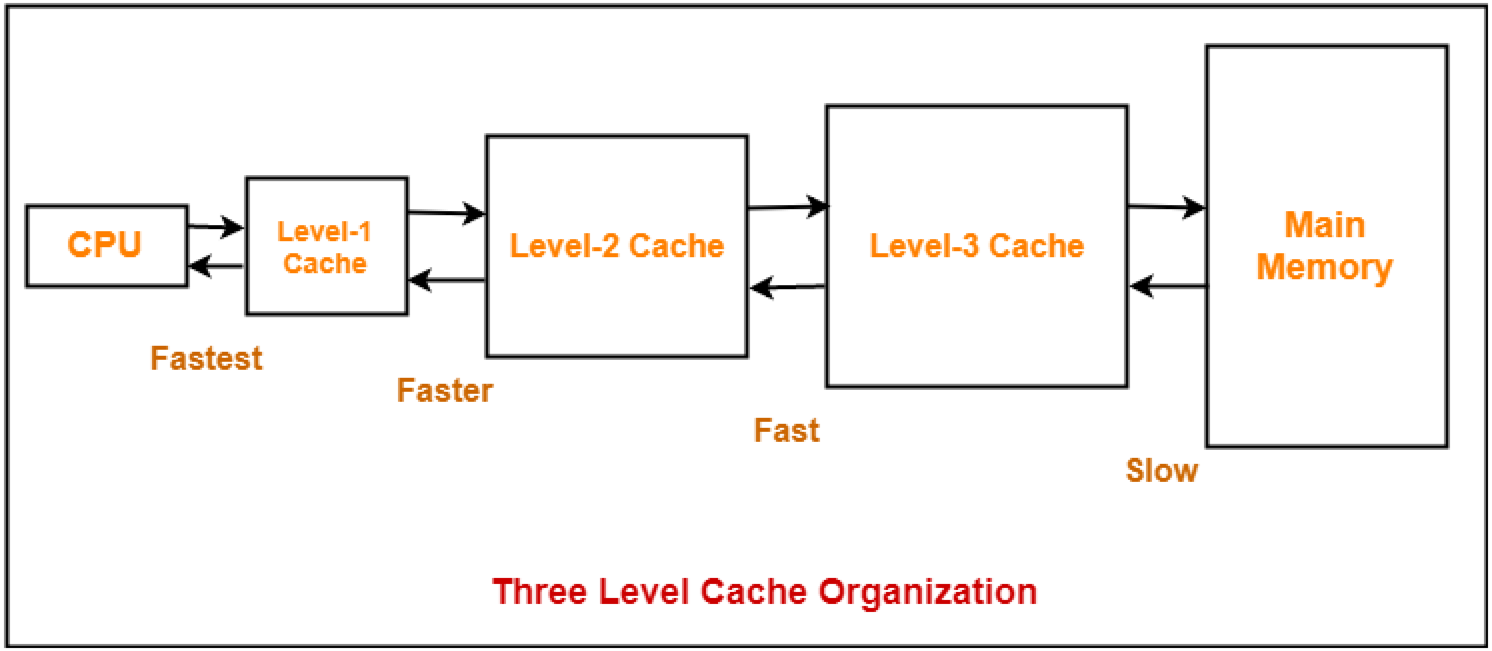

MOTHERBOARD
Main memory chips
Processor chip
I/O chips
PROCESSOR CHIP
Core
Core
Core
Core
L3 cache
L3 cache
Core
Core
Core
Core
CORE
Instruction logic
Arithmetic and logic unit (ALU)
Load/ store logic
L1 I-cache
L1 data cache
L2 instruction cache
L2 data cache

Simplified View of Major Elements of a Multicore Computer

Lets look at

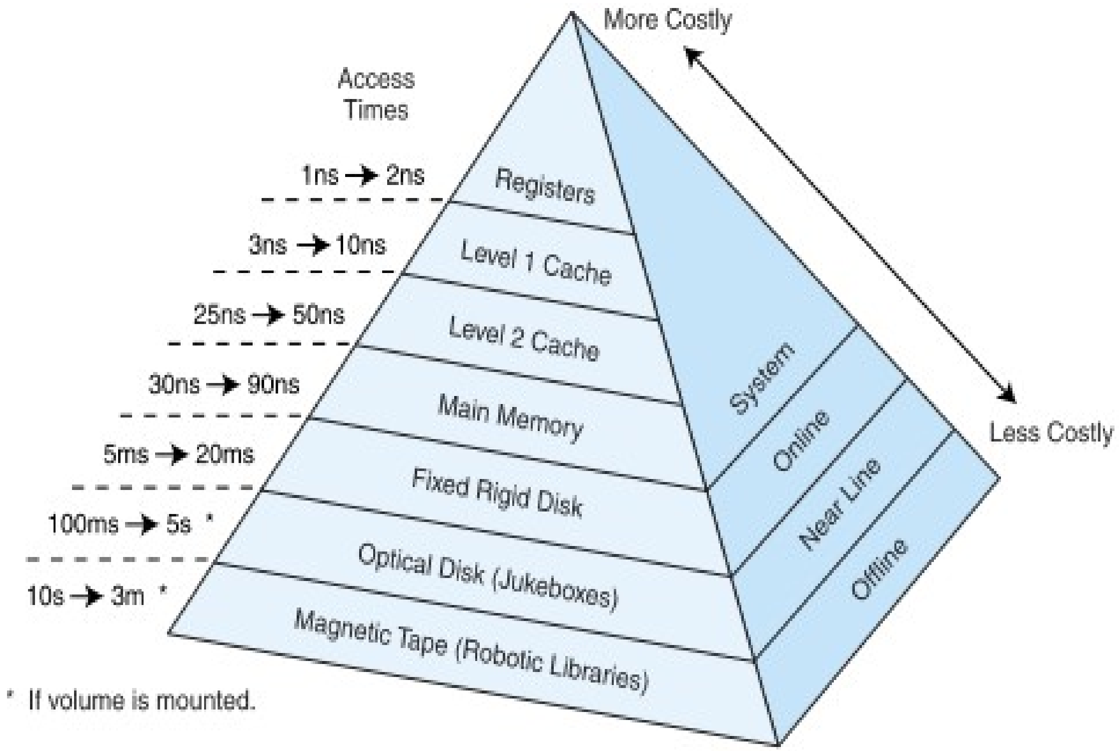
- Chip
- Mother board
- Memory, I/O controller, and other components
- Expansion slots
- Chips -> Transistors
- Multi level cache L1-L2-L3 Cache

Simplified View of Major Elements of a Multicore Computer


CPU
Level-1 Cache
Level-2 Cache
Level-3 Cache
Main Memory
Fastest
Faster
Fast
Slow

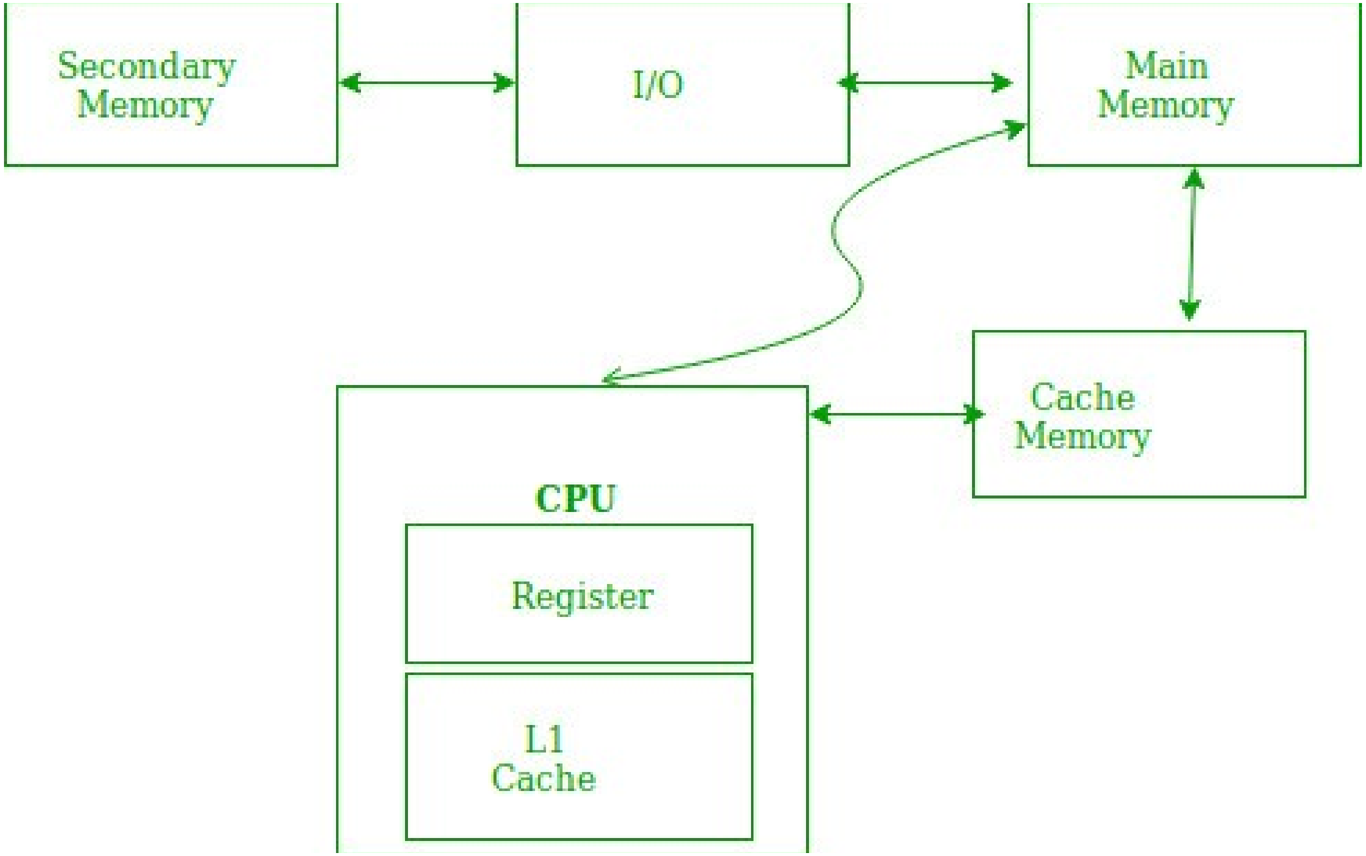
Three Level Cache Organization

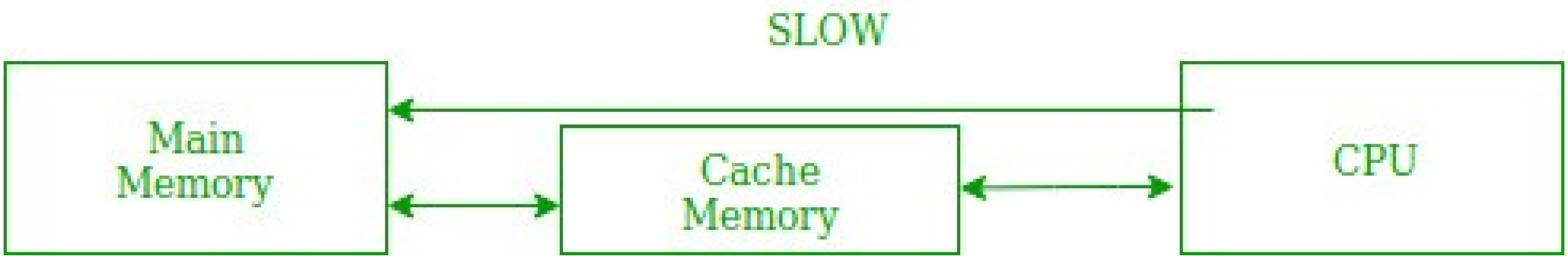
Computer Memory Hierarchy

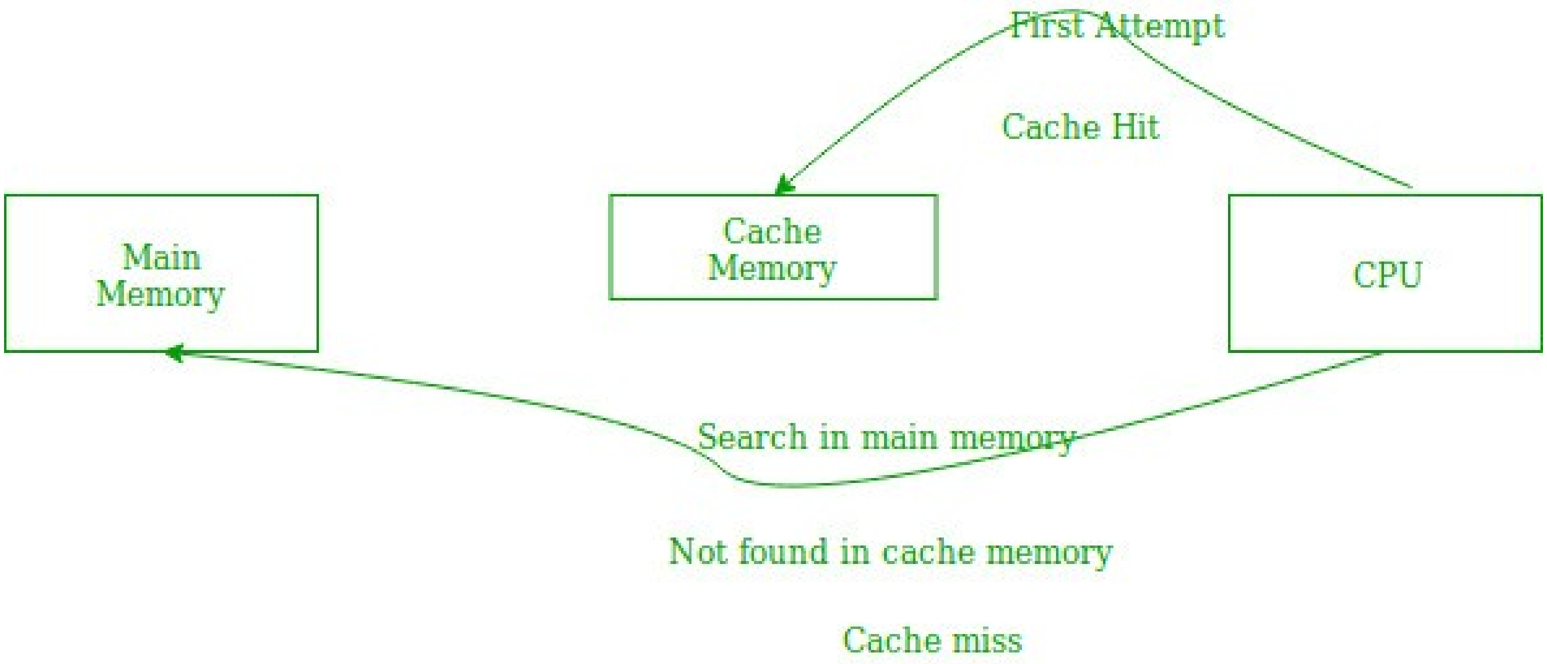


More Costly
Access Times
1ns → 2ns
3ns → 10ns
25ns → 50ns
30ns → 90ns
5ms → 20ms
100ms → 5s *
10s → 3m *
Registers
Level 1 Cache
Level 2 Cache
Main Memory
Fixed Rigid Disk
Optical Disk (Jukeboxes)
Magnetic Tape (Robotic Libraries)
System
Online
Near Line
Offline
Less Costly
* If volume is mounted.

Source: Null, Linda and Lobur, Julia (2003). *Computer Organization and Architecture* (p. 236). Sudbury, MA: Jones and Bartlett Publishers


Secondary Memory
I/O
Main Memory
Cache Memory
CPU
Register
L1 Cache


SLOW
Main Memory
Cache Memory
CPU



First Attempt

Cache Hit

Main Memory

Cache Memory

CPU

Search in main memory

Not found in cache memory

Cache miss

Simplified View of Major Elements of a Multicore Computer

- L1 cache, split between an instruction cache (I-cache) that is used for the transfer of instructions to and from main memory, and an L1 data cache, for the transfer of operands and results.
- L2 cache as part of the core. In many cases, this cache is also split between instruction and data caches, although a combined, single L2 cache is also used.

Simplified View of Major Elements of a Multicore Computer

In general terms, the functional elements of a core are:

■ Instruction logic: This includes the tasks involved in fetching instructions, and decoding each instruction to determine the instruction operation and the memory locations of any operands.

■ Arithmetic and logic unit (ALU): Performs the operation specified by an instruction.

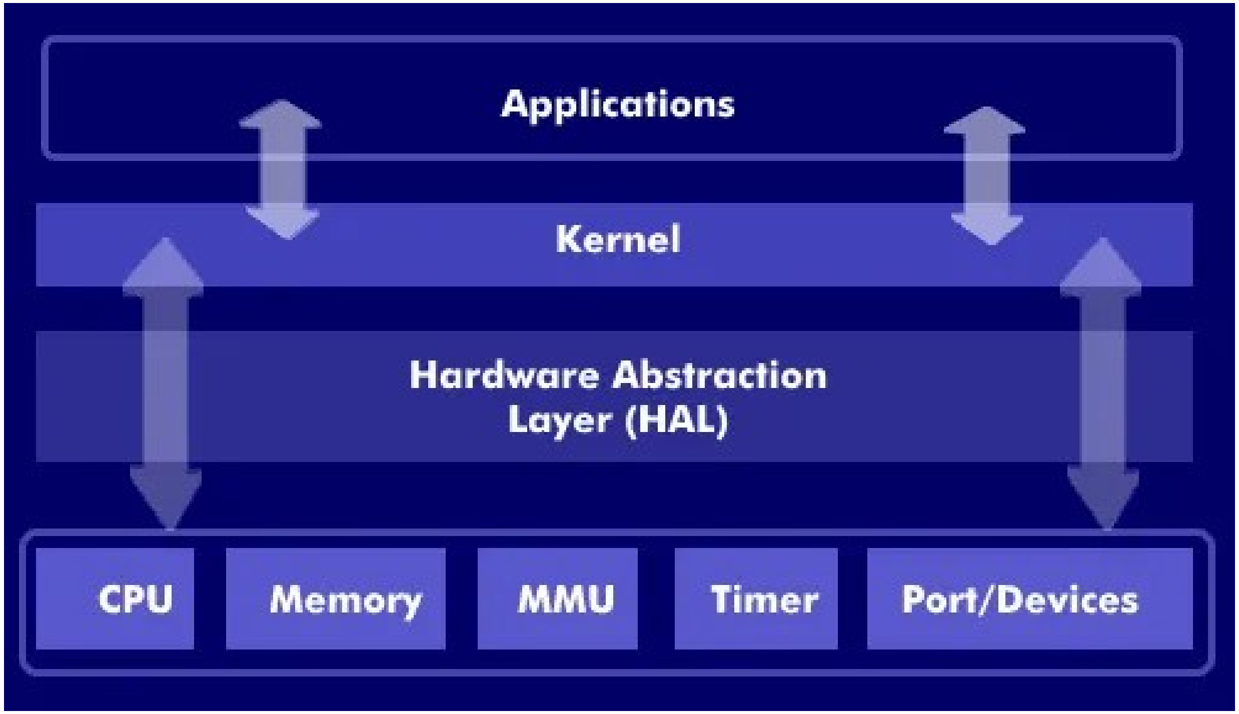
■ Load/store logic: Manages the transfer of data to and from main memory via cache.

innovate achieve lead

HARDWARE ABSTRACTION VS SOFTWARE ABSTRACTION

- **Hardware abstractions** are sets of routines in software that provide programs with access to hardware resources through programming interfaces.
- Hardware abstractions often allow programmers to write device-independent, high performance applications by providing standard operating system (OS) calls to hardware.
- **Hardware abstraction layer** is a thin layer of software at the base of the operating system that offers a uniform interface between the underlying hardware and the higher layers of the operating system, hiding hardware differences from those higher layers.

Hardware Abstraction Layer Example



Applications

Kernel

Hardware Abstraction Layer (HAL)

CPU

Memory

MMU

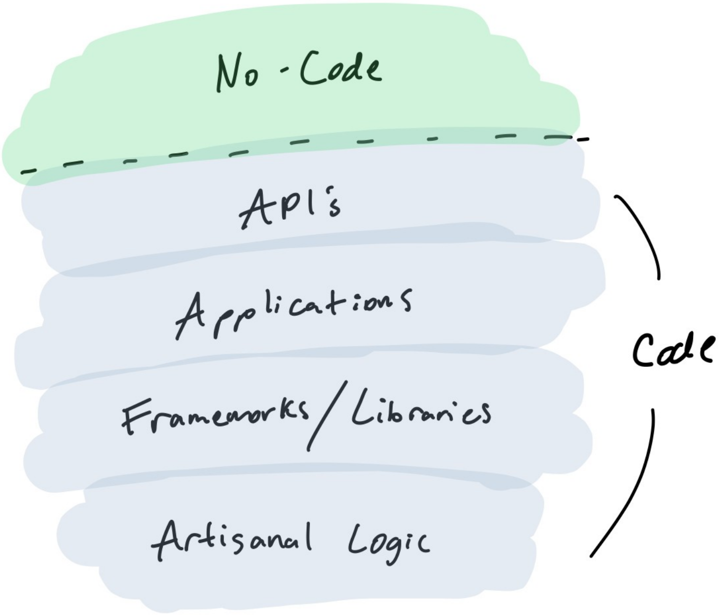
Timer

Port/Devices

Software abstraction

- The software abstraction is the creation of abstract concept-objects by mirroring common features or attributes of various non-abstract objects or systems of study.
- Examples of software models that use layers of abstraction include the OSI model for network protocols, OpenGL and other graphics libraries.

Modern day example

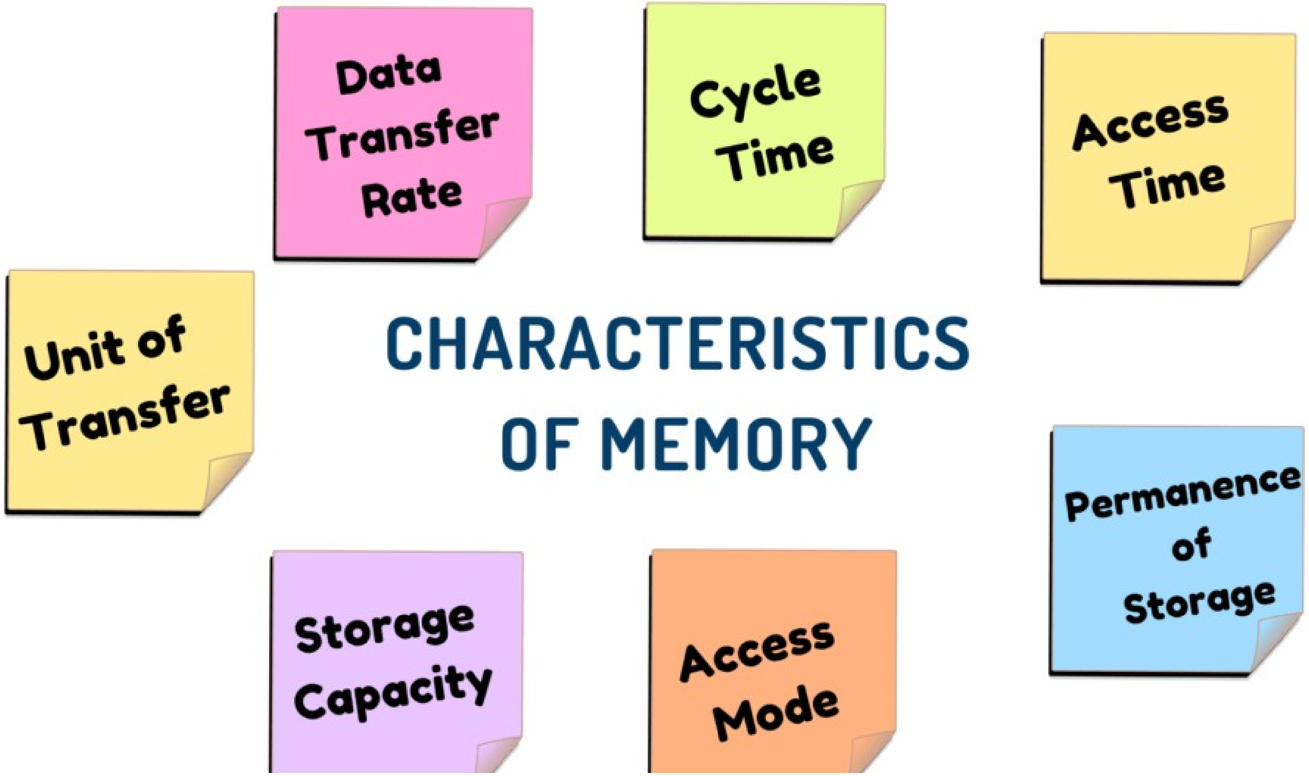

No - Code
API's
Applications
Frameworks / Libraries
Artisanal Logic
Code

THE THREE FUNDAMENTAL ABSTRACTIONS

The three fundamental Abstractions

1. Memory
2. Interpreters
3. Communication links

THE THREE FUNDAMENTAL ABSTRACTIONS - Memory

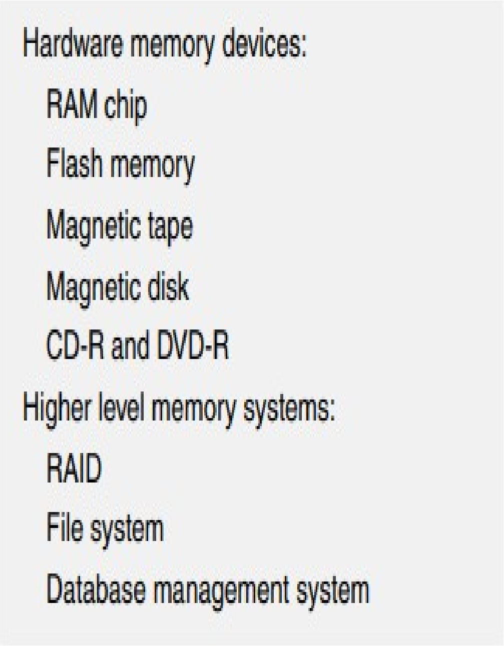

CHARACTERISTICS OF MEMORY
Data Transfer Rate
Cycle Time
Access Time
Unit of Transfer
Permanence of Storage
Storage Capacity
Access Mode

THE THREE FUNDAMENTAL ABSTRACTIONS - Memory

- Memory, sometimes called storage, is the system component that remembers data values for use in computation.
- All memory devices fit a simple abstract model that has two operations, named write and read:
  - ***WRITE (name, value)***
  - ***Value ← READ (name)***

THE THREE FUNDAMENTAL ABSTRACTIONS - Memory

- Memories can be either volatile or non-volatile.
    - Volatile - Consumes energy to retain information
    - Non Volatile – retains its content, and when power is again available, READ operations return the same values as before.
    - By connecting a volatile memory to a battery or an uninterruptible power supply, it can be made durable, which means that it is designed to remember things for at least some specified period, known as its durability.
    - Even non-volatile memory devices are subject to eventual deterioration, known as decay, so they usually also have a specified durability, perhaps measured in years



Hardware memory devices:
  RAM chip
  Flash memory
  Magnetic tape
  Magnetic disk
  CD-R and DVD-R
Higher level memory systems:
  RAID
  File system
  Database management system

Read/Write Coherence & Atomicity

- Two useful properties for a memory are read/write coherence and before-or-after atomicity.
  - Result of the READ of a named cell is always the same as the most recent write to that cell.
- Before-or-after atomicity: Irrespective of before or after operations
- There are a surprising number of threats to read/write coherence and before-or-after atomicity:
  - **Concurrency**
  - **Remote storage**
  - **Performance enhancements**
  - **Cell size incommensurate with value size**
  - **Replicated storage**

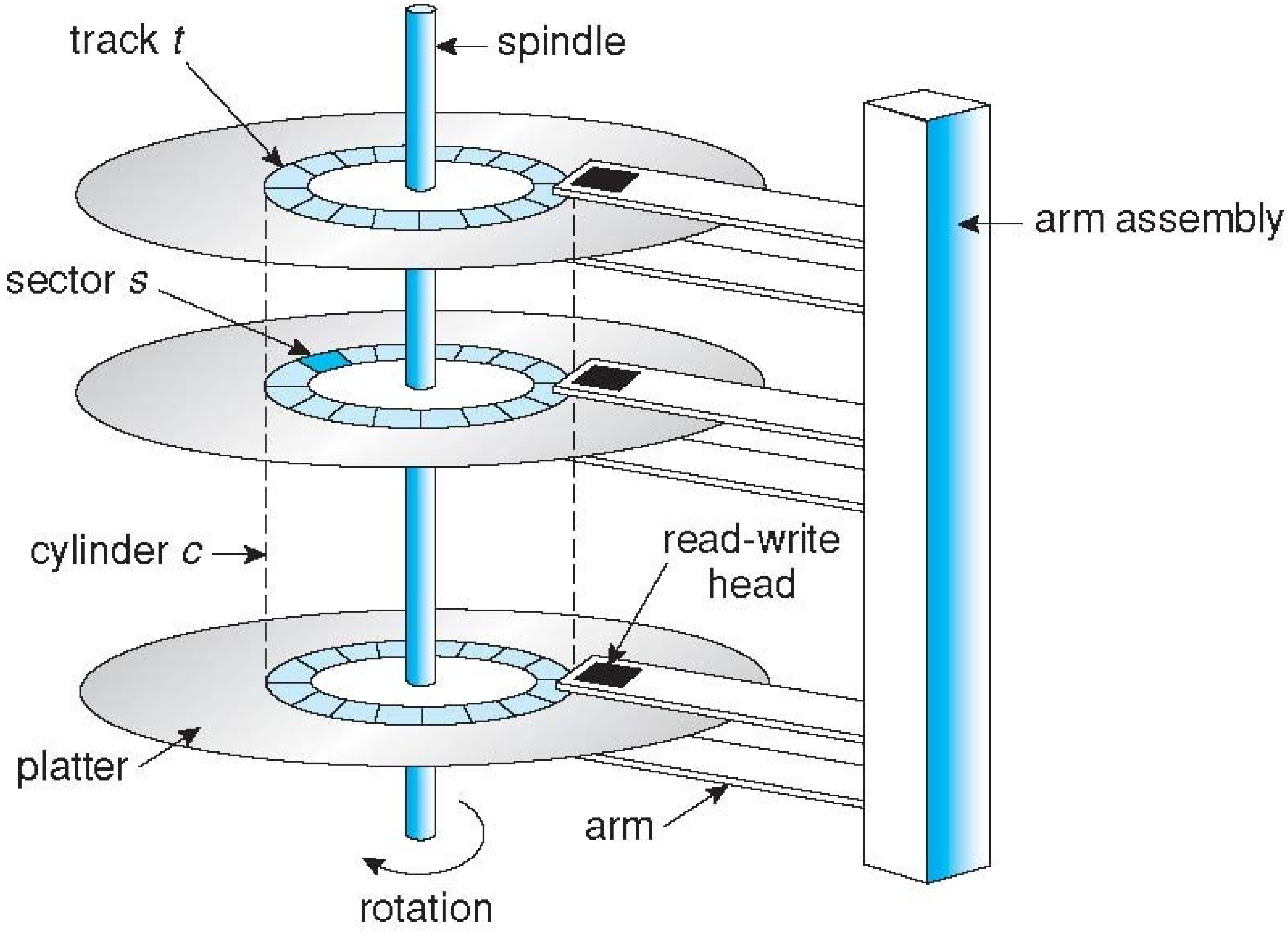
Memory Latency

- An important property of a memory is the time it takes for a read or a write to complete, which is known as its latency (often called access time).
- In the magnetic disk memory the latency of a particular sector depends on the mechanical state of the device at the instant the user requests access.
- A random access memory (RAM) is one for which the latency for memory cells chosen at random is approximately the same as the latency for cells chosen in the pattern best suited for that memory device.
- An electronic memory chip is usually configured for random access.

Overview of Mass Storage Structure

- **Magnetic disks** provide bulk of secondary storage of modern computers
  - Drives rotate at 60 to 200 times per second
  - **Transfer rate** is rate at which data flow between drive and computer
  - **Positioning time** (**random-access time**) is time to move disk arm to desired cylinder (**seek time**) and time for desired sector to rotate under the disk head (**rotational latency**)
  - **Head crash** results from disk head making contact with the disk surface
    - That's bad
- Disks can be removable
- Drive attached to computer via **I/O bus**
  - Busses vary, including **EIDE**, **ATA**, **SATA**, **USB**, **Fibre Channel**, **SCSI**
  - **Host controller** in computer uses bus to talk to **disk**

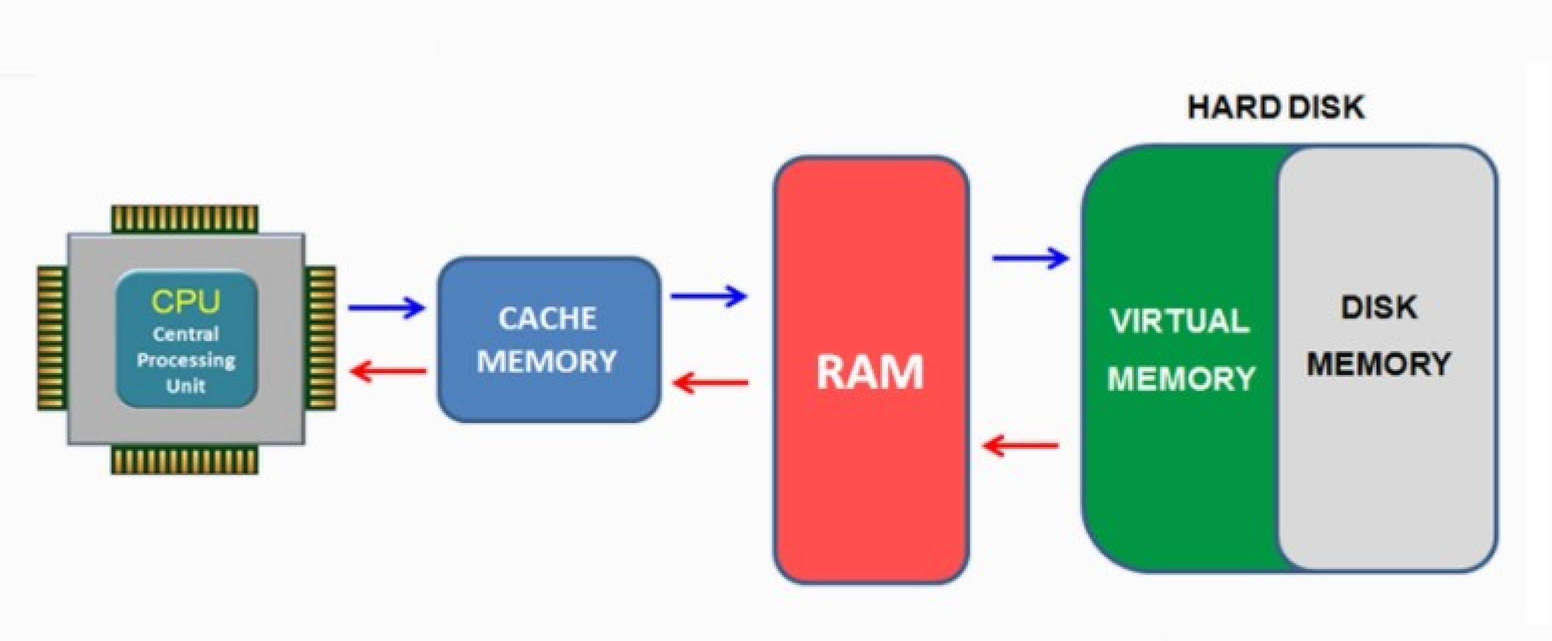
Moving-head Disk Mechanism


track *t*
spindle
arm assembly
sector *s*
cylinder *c*
read-write head
platter
arm
rotation

Disk Scheduling

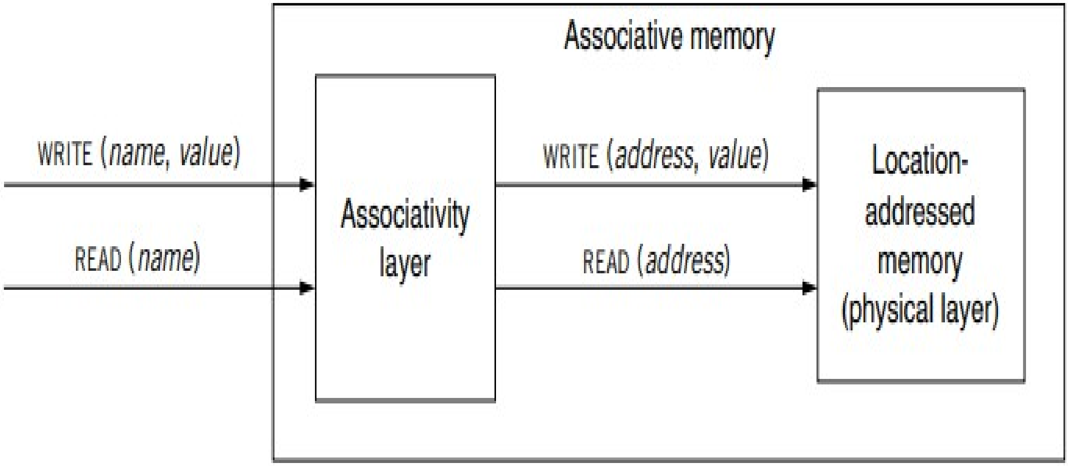
- The operating system is responsible for using hardware efficiently — for the disk drives, this means having a fast access time and disk bandwidth
- Access time has two major components
  - **Seek time** is the time for the disk are to move the heads to the cylinder containing the desired sector
  - **Rotational latency** is the additional time waiting for the disk to rotate the desired sector to the disk head
- Minimize seek time
- Seek time ≈ seek distance
- Disk **bandwidth** is the total number of bytes transferred, divided by the total time between the

Memory names and addresses


HARD DISK
CPU
Central
Processing
Unit
CACHE
MEMORY
RAM
VIRTUAL
MEMORY
DISK
MEMORY

Memory Names and Addresses

- Physical implementations of memory devices nearly always name a memory cell by the geometric coordinates of its physical storage location.
- Thus, for example, an electronic memory chip is organized as a two-dimensional array of flip-flops, each holding one named bit.
- A memory system that accepts unconstrained names is called an associative memory.
- Since physical memories are generally location-addressed, a designer creates an associative memory by interposing an associativity layer, which may be implemented either with hardware or software that maps unconstrained higher-level names to the constrained integer names of an underlying location-addressed memory.


Associative memory
WRITE (*name*, *value*)
READ (*name*)
Associativity layer
WRITE (*address*, *value*)
READ (*address*)
Location-addressed memory (physical layer)

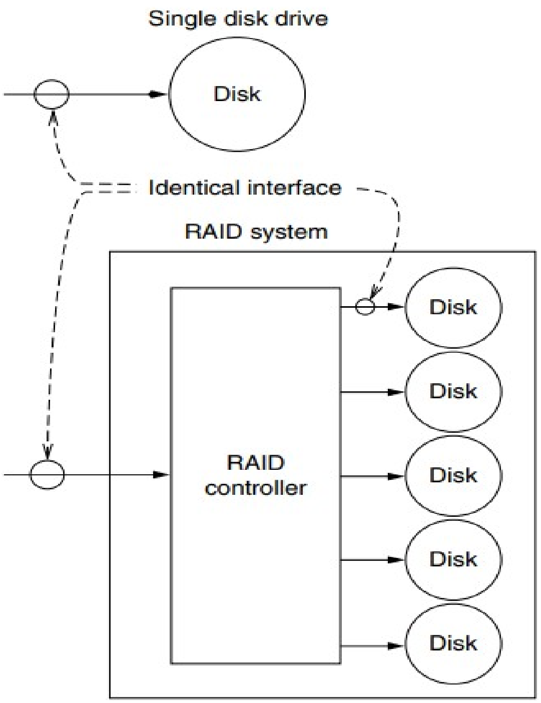
Exploiting the Memory Abstraction: RAID

https://www.youtube.com/watch?v=U-OCdTeZLac

- RAID is an acronym for Redundant Array of Independent (or Inexpensive) Disks.
- A RAID system consists of a set of disk drives and a controller configured with an electrical and programming interface that is identical to the interface of a single disk drive, as shown in Figure.
- The RAID controller intercepts read and write requests coming across its interface, and it directs them to one or more of the disks. RAID has two distinct goals:

▪ *Improved performance, by reading or writing disks concurrently*

▪ *Improved durability, by writing information on more than one disk*


Single disk drive
Disk
Identical interface
RAID system
RAID controller
Disk
Disk
Disk
Disk
Disk

Abstraction II - Interpreters

- Interpreters are the active elements of a computer system; they perform the actions that constitute computations.
- Figure lists some examples of interpreters that may be familiar. As with memory, interpreters also come in a wide range of physical manifestations.

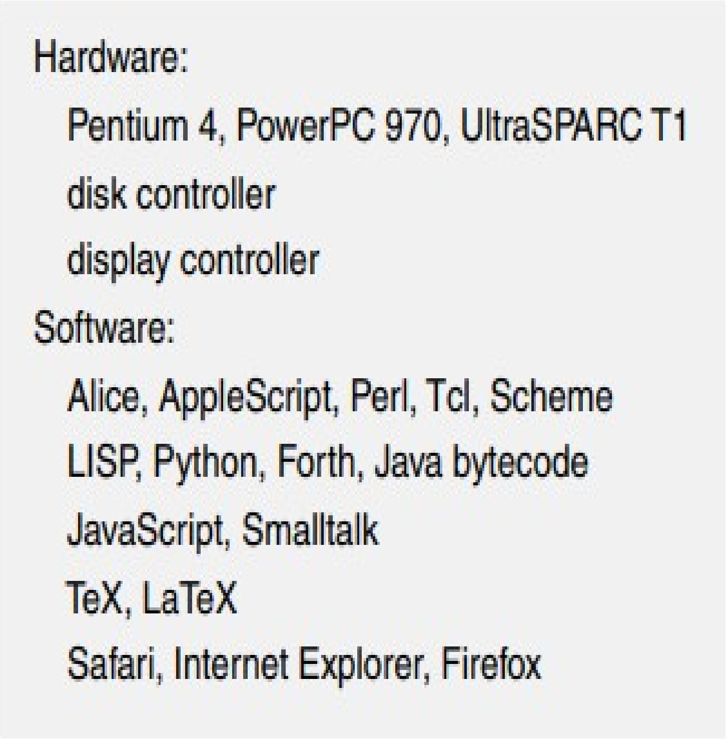
Abstraction II - Interpreters

- However, they too can be described with a simple abstraction, consisting of just three components:

1. An instruction reference, which tells the interpreter where to find its next instruction

2. A repertoire, which defines the set of actions the interpreter is prepared to perform when it retrieves an instruction from the location named by the instruction reference

3. An environment reference, which tells the interpreter where to find its environment, the current state on which the interpreter should perform the action of the current instruction.

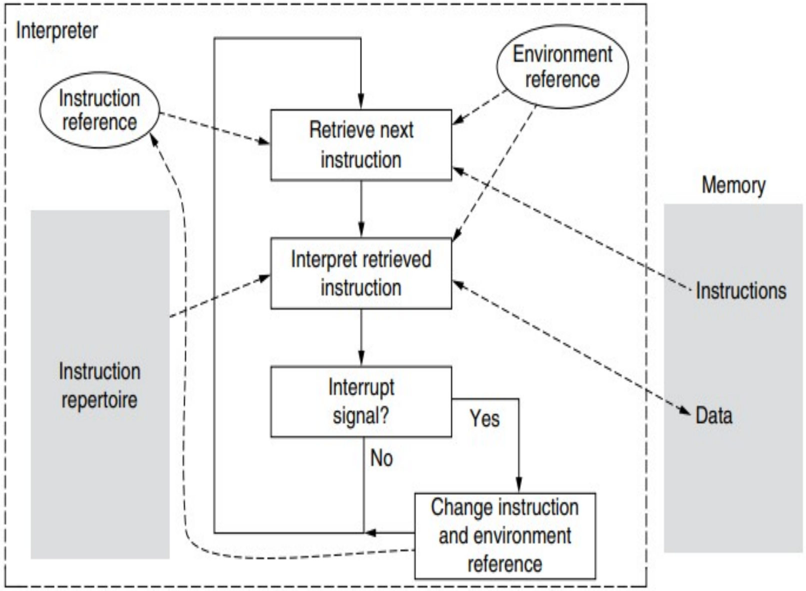


Hardware:

- Pentium 4, PowerPC 970, UltraSPARC T1
- disk controller
- display controller

Software:

- Alice, AppleScript, Perl, Tcl, Scheme
- LISP, Python, Forth, Java bytecode
- JavaScript, Smalltalk
- TeX, LaTeX
- Safari, Internet Explorer, Firefox


Interpreter
Instruction reference
Environment reference
Retrieve next instruction
Interpret retrieved instruction
Interrupt signal?
Yes
No
Change instruction and environment reference
Instruction repertoire
Memory
Instructions
Data

```
procedure INTERPRET()
    do forever
        instruction ← READ (instruction_reference)
        perform instruction in the context of environment_reference
        if interrupt_signal = TRUE then
            instruction_reference ← entry point of INTERRUPT_HANDLER
            environment_reference ← environment ref of INTERRUPT_HANDLER
```

1) Where is the instruction – Sequential or non-sequential
2) Where are the operands
3) Interrupts

Processors

- The processor’s instruction reference is a program counter, stored in a fast memory register inside the processor.
- The program counter contains the address of the memory location that stores the next instruction of the current program.
- The environment reference of the processor consists in part of a small amount of built-in location-addressed memory in the form of named (by number) registers for fast access to temporary results of computations.
- The processor also implements interrupts. An interrupt can occur because the processor has detected some problem with the running program.
- An interrupt can also occur because a signal arrives from outside the processor, indicating that some external device needs attention.

Communication Links

- A communication link provides a way for information to move between physically separated components.

- Programs that invoke send and receive must take these different semantics explicitly into account.
- On the other hand, some communication link implementations do provide a layer that does its best to hide a send/receive interface behind a read/write interface.
- Just as with memory and interpreters, designers organize and implement communication links in layers.

ORGANIZING COMPUTER SYSTEMS WITH NAMES AND LAYERS

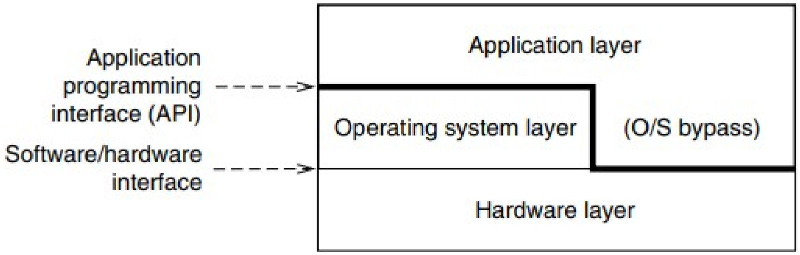
The bottom laver consists of hardware components, such as processors, memories, and communication links.

The middle laver consists of a collection of software modules, called the operating system,

The top laver consists of software that implements application specific functions,

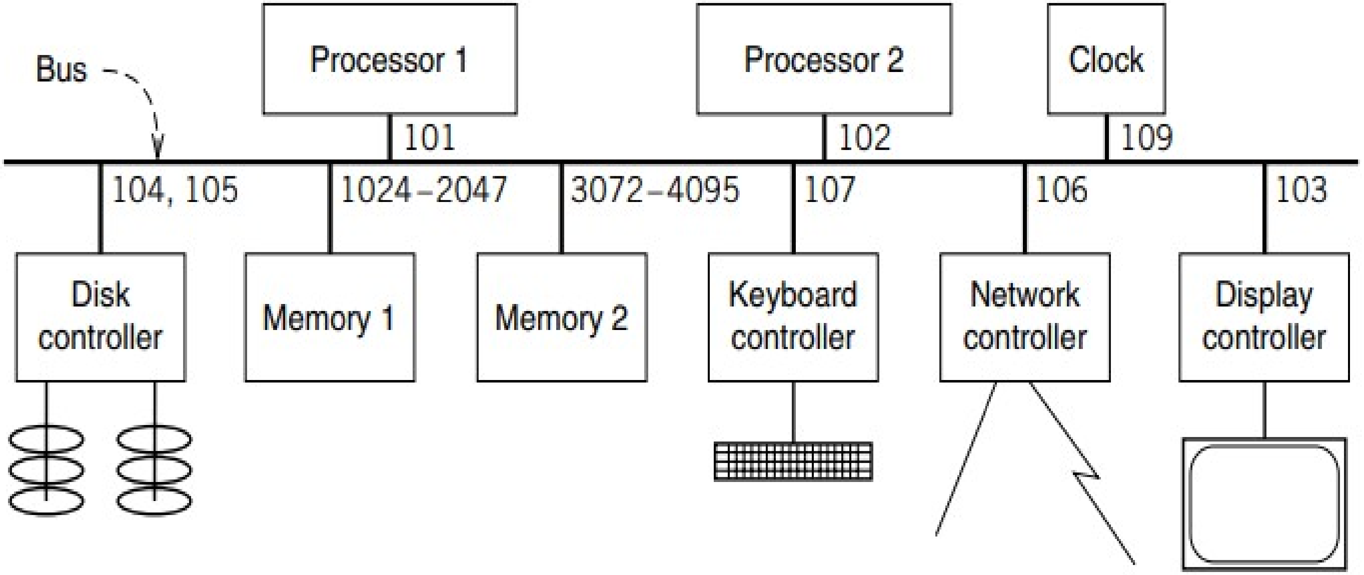
In principle, every software module can be implemented in hardware.

Similarly, most hardware modules can also be implemented in software, except for a few foundational components such as transistors and wires.


Application programming interface (API)
Software/hardware interface
Application layer
Operating system layer
(O/S bypass)
Hardware layer

A Hardware Layer: The Bus

- The processor modules interpret programs,
  - the random access memory modules store both programs and data,
  - and the input/output (I/O) modules implement communication links to the world outside the computer.
- The I/O modules turn out to be specialized interpreters that implement I/O programs.

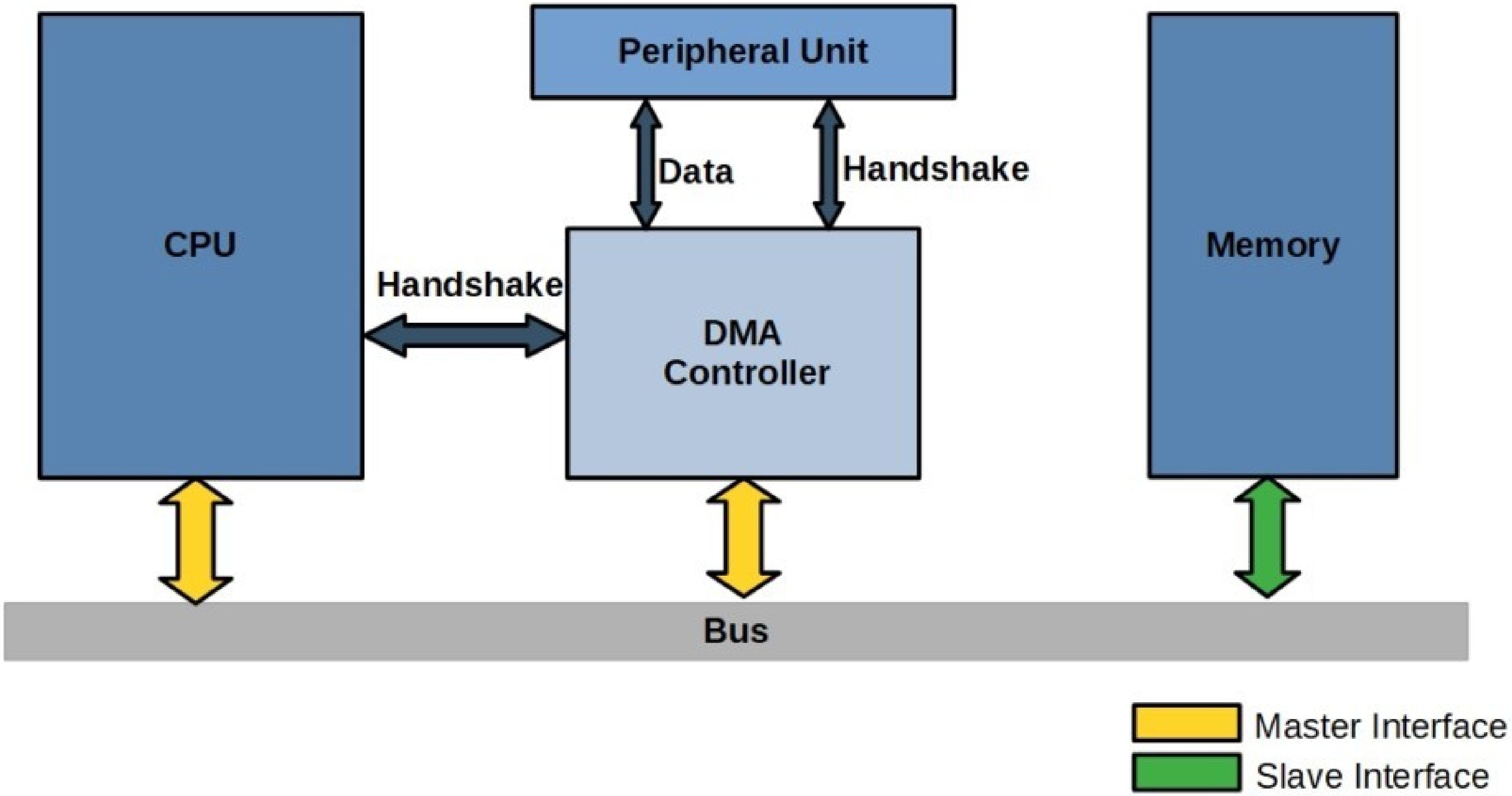

Bus
Processor 1
101
Processor 2
102
Clock
109
104, 105
1024–2047
3072–4095
107
106
103
Disk controller
Memory 1
Memory 2
Keyboard controller
Network controller
Display controller

A Hardware Layer: The Bus

- Any bus module that wishes to send a message over the bus must know a bus address that the intended recipient is configured to accept.
-
- For example, suppose that processor #2, while interpreting a running application program, encounters the instruction

  LOAD 1742, R1
- which means “load the contents of memory address 1742 into processor register R1”.
- In the simplest scheme, the processor just translates addresses it finds in instructions directly to bus addresses without change.

A Hardware Layer: The Bus


Peripheral Unit
Data
Handshake
CPU
Handshake
DMA Controller
Memory
Bus
Master Interface
Slave Interface

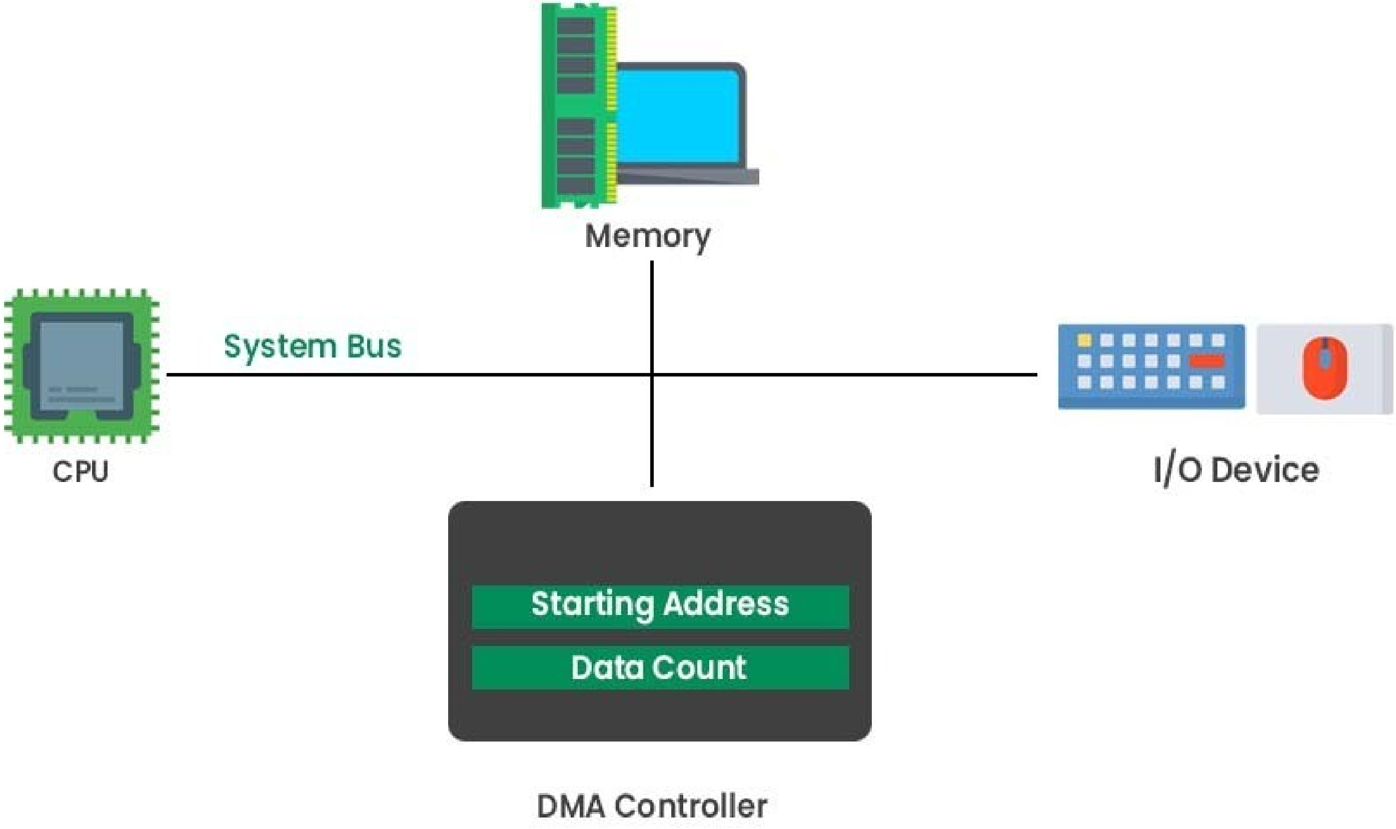
A Hardware Layer: The Bus

- DMA- Upgrading the disk controller to use a technique called direct memory access, or DMA.
  - With this technique, when a processor sends a request to a disk controller to read a block of data from the disk, it includes the address of a buffer in memory as a field of the request message.
  - Then, as data streams in from the disk, the disk controller sends it directly to the memory module, incrementing the memory address appropriately between sends.
  - In addition to relieving the load on the processor, DMA also reduces the load on the shared bus because it transfers each piece of data across the bus just once (from the disk controller to the memory) rather than twice (first from the disk controller to the processor and then from the processor to the memory).

- DMA basically stands for Direct Memory Access.
- It is a process which enables data transfer between the Memory and the IO (Input/ Output) device without the need of or you can say without the involvement of CPU during data transfer.

Working of DMA

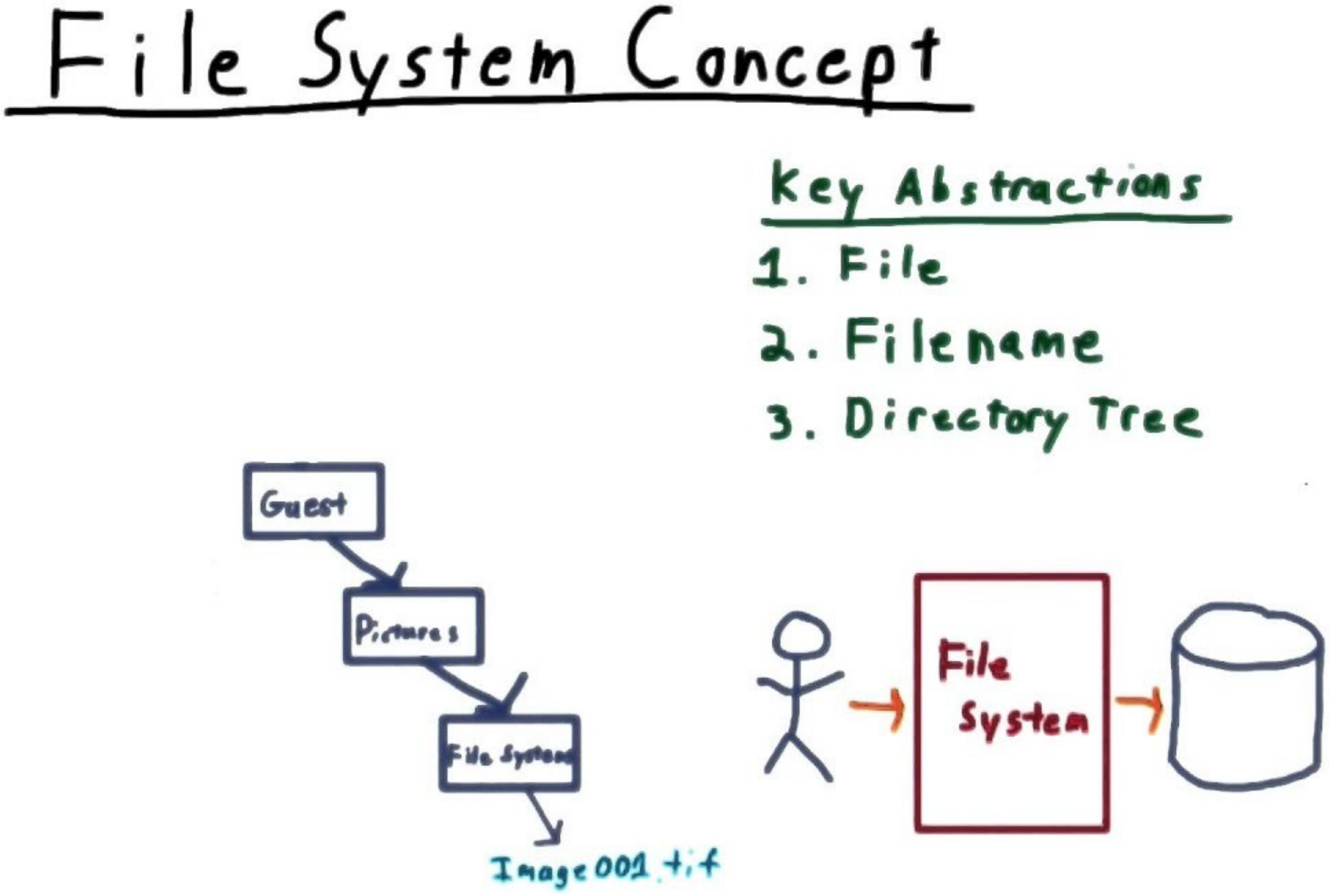
- For DMA, you basically need a hardware called DMAC (Direct Memory Access Controller) which will help in the throughout process of data transfer between the Memory and IO device directly.
- First what happens is IO device sends the DMA request to DMA Controller, then further DMAC device sends HOLD signal to CPU by which it asks CPU for several information which are needed while transferring data.
- CPU then shares two basic information with DMAC before the Data transfer which are: Starting address (memory address starting from where data transfer should be performed) and Data Count (no of bytes or words to be transferred).
- CPU then sends HLDACK (Hold Acknowledgement) back to DMAC illustrating that now DMAC can successfully pass on the information.
- Then further DMAC shares the DMA ACK (DMA Acknowledgement) to the IO device which would eventually let IO device to access or transfer the data from memory in a direct and efficient manner.


Memory
System Bus
CPU
I/O Device
Starting Address
Data Count
DMA Controller

A Software Layer

- The File Abstraction
  - The middle and higher layers of a computer system are usually implemented as software modules.
  - To make this layered organization concrete, consider the file, a high-level version of the memory abstraction.
- A file holds an array of bits or bytes, the number of which the application chooses. A file has two key properties:
  - It is durable
  - It has a name

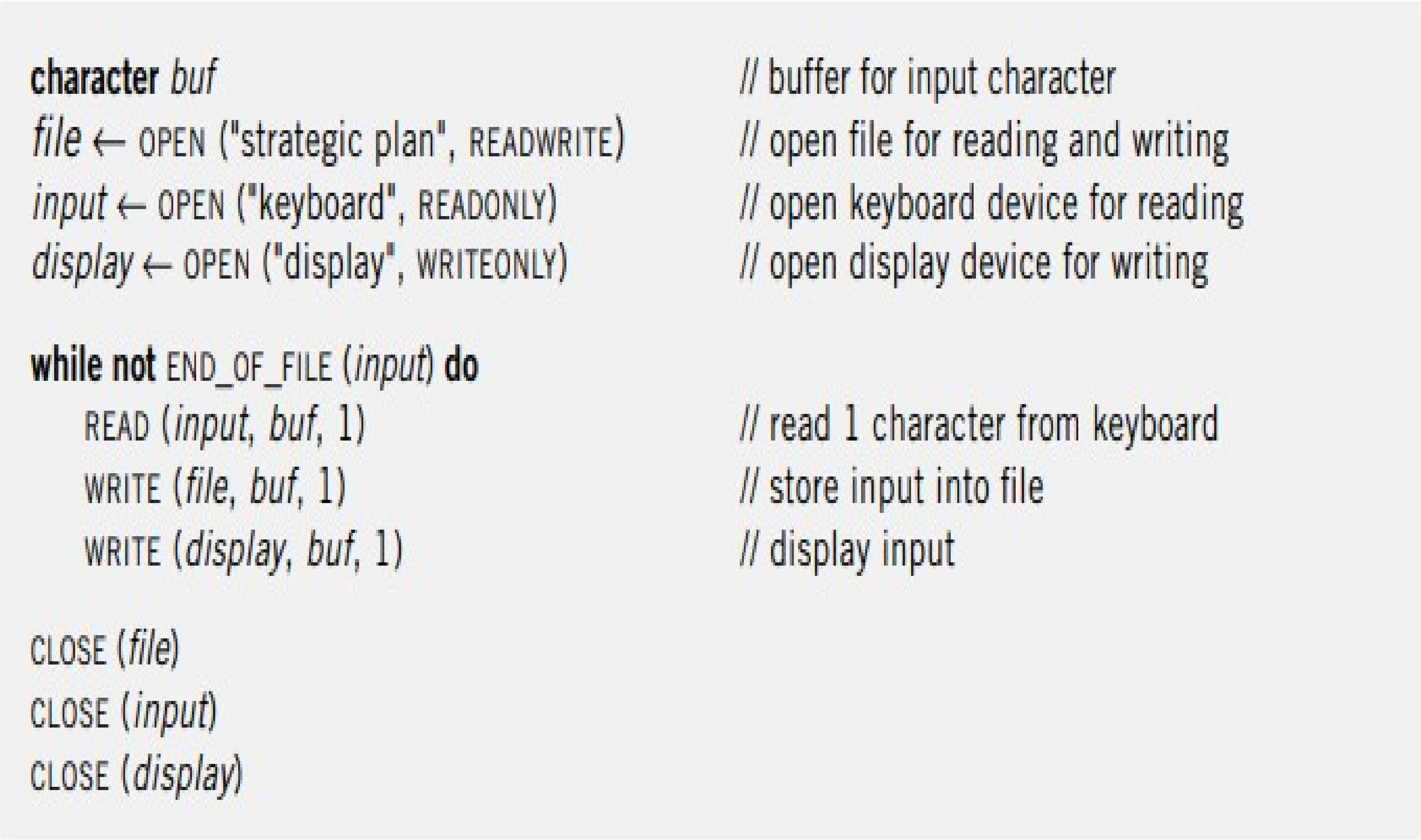
A Software Layer


File System Concept
Key Abstractions
1. File
2. Filename
3. Directory Tree
Guest
Pictures
File System
Image001.tif
File System

A Software Layer

- Taken together, these two features mean that if, for example, Alice creates a new file named “strategic plan”, writes some information in it, shuts down the computer, and the next day turns it on again, she will then be able to read the file named “strategic plan” and get back its content.
- Furthermore, she can tell Bob to look at the file named “strategic plan”.
- When Bob asks the system to read a file with that name, he will read the file that she created.
- Most file systems also provide other additional properties for files, such as timestamps to determine when they were created, last modified, or last used, assurances about their durability, and the ability to control who may share them.

Software layer - Files



```
character buf                                        // buffer for input character
file ← OPEN ("strategic plan", READWRITE)            // open file for reading and writing
input ← OPEN ("keyboard", READONLY)                  // open keyboard device for reading
display ← OPEN ("display", WRITEONLY)                // open display device for writing

while not END_OF_FILE (input) do
    READ (input, buf, 1)                             // read 1 character from keyboard
    WRITE (file, buf, 1)                             // store input into file
    WRITE (display, buf, 1)                          // display input

CLOSE (file)
CLOSE (input)
CLOSE (display)
```

Software layer - Files

- The system layer implements files using modules from the hardware layer.
  - A typical API for the file abstraction contains calls to open a file, to read and write parts of the file, and to close the file.
  - The open call translates the file name into a temporary name in a local name space to be used by the read and write operations.
  - Also, open usually checks whether this user is permitted access to the file. As its last step, open sets a cursor, sometimes called a file pointer, to zero.

Software layer - files

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
   int num;
   FILE *fptr;
   fptr = fopen("C:\\program.txt","w");

   if(fptr == NULL)
   {
      printf("Error!");
      exit(1);
   }

   printf("Enter num: ");
   scanf("%d",&num);

   fprintf(fptr,"%d",num);
   fclose(fptr);

   return 0;
}
```

Software layer - files

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
   int num;
   FILE *fptr;

   if ((fptr = fopen("C:\\program.txt","r")) == NULL){
       printf("Error! opening file");

       // Program exits if the file pointer returns NULL.
       exit(1);
   }

   fscanf(fptr,"%d", &num);

   printf("Value of n=%d", num);
   fclose(fptr);

   return 0;
}
```

Software layer - files

- The cursor records an offset from the beginning of the file to be used as the starting point for reads and writes.
- A call to read delivers to the caller a specified number of bytes from the file, starting from the read cursor.
- It also adds to the read cursor the number of bytes read so that the next read proceeds where the previous read left off.
- If the program asks to read bytes beyond the end of the file, read returns some kind of end-of-file status indicator.

Software layer - files

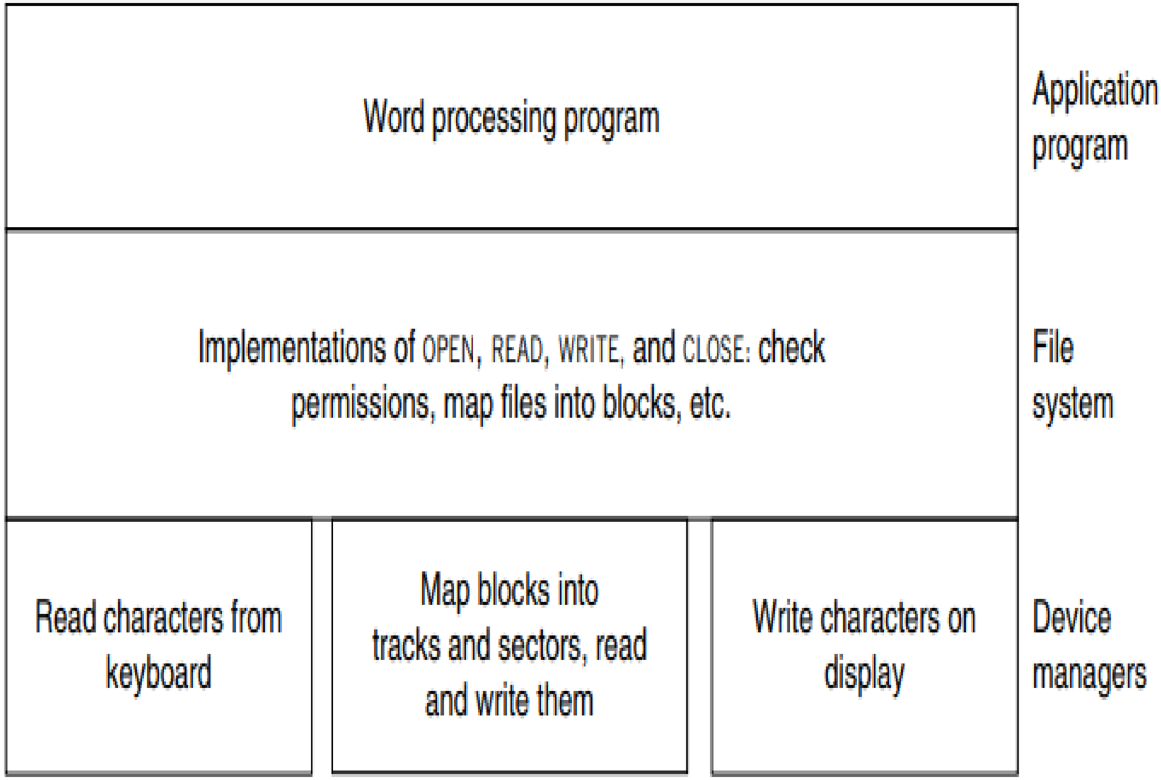
- If there is not enough space on the device to write that many bytes, the write procedure fails by returning some kind of device-full error status or exception.
- Finally, when the program is finished reading and writing, it calls the close procedure. Close frees up any internal state that the file system maintains for the file.
- The file system module implements the file API by mapping bytes of the file to disk sectors.
- CLOSE was introduced to simplify resource management: when an application invokes close, the file system knows that the application doesn’t need the resources (e.g., the cursor) that the file system maintains internally.

Software layer - files

- The file is such a convenient memory abstraction that in some systems every input/output device in a computer system provides a file interface

- In such systems, files not only are an abstraction for non-volatile memories (e.g., magnetic disks), but they are also a convenient interface to the keyboard device, the display, communication links, and so on.
- In such systems, each I/O device has a name in the file naming scheme.
- A program opens the keyboard device, reads bytes from the keyboard device, and then closes the keyboard device, without having to know any details about the keyboard management procedure, what type of keyboard it is, and the like.
- Similarly, to interact with the display, a program can open the display device, write to it, and close it. The program need not know any details about the display.

Software layer - files



Word processing program

Implementations of OPEN, READ, WRITE, and CLOSE: check permissions, map files into blocks, etc.

Read characters from keyboard

Map blocks into tracks and sectors, read and write them

Write characters on display

Application program

File system

Device managers

THANK YOU

BITS Pilani
Pilani Campus

BITS Pilani
Pilani Campus

COMPUTING AND DESIGN

SESSION 3

Course design by - Dr. Lucy J. Gudino &
Dr Chandrasekhar
Faculty – Thangakumar J
WILP & Department of CS & IS

Last Session

List of Topic Title	Text/Ref Book/external resource
• Major Modules of a Typical Computer System: Processor(s), Memory, I/O and Storage Devices • Hardware abstraction vs. Software • The three fundamental Abstractions • Organizing Computer Systems	R6 (1.2) T2 (2.1 and 2.3)

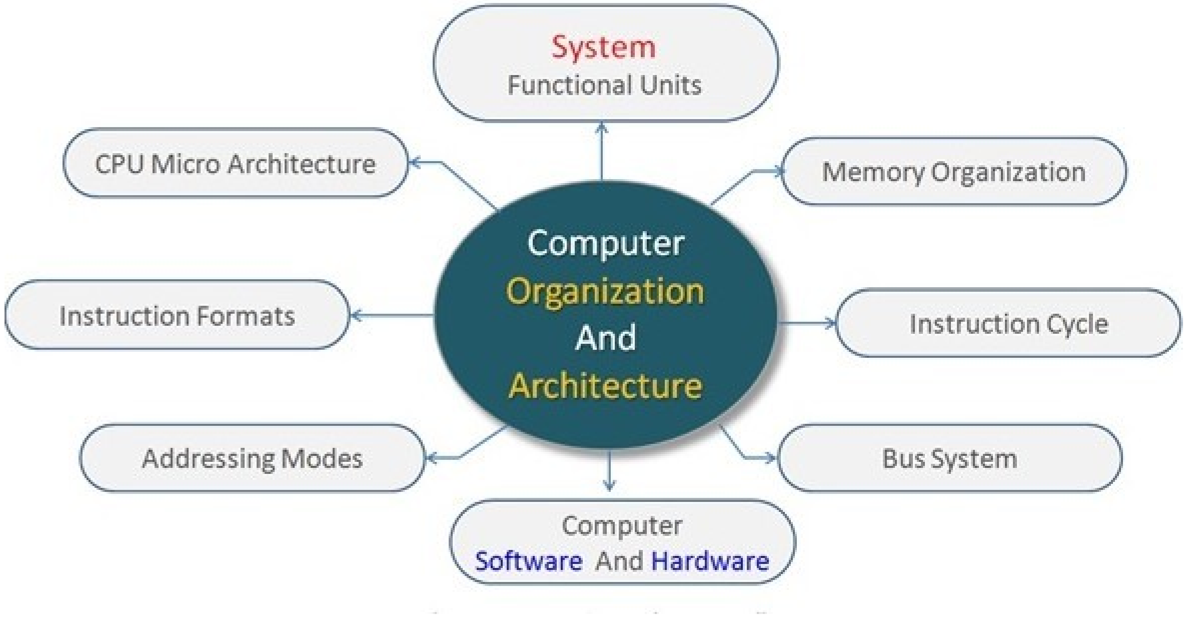
ISA as the Interface

Today's topic

- Computer Architecture Vs. Organization (Abstraction vs. Realization)
- Von-Neumann Architecture vs Harvard Architecture
- Features and Components of an ISA.
  - CISC Vs. RISC
- Design Goals

Computer Architecture VS Organization (Abstraction vs Realization)

- Computer architecture refers to those attributes of a system visible to a programmer or, put another way, those attributes that have a direct impact on the logical execution of a program.



System
Functional Units

CPU Micro Architecture

Memory Organization

Computer
Organization
And
Architecture

Instruction Formats

Instruction Cycle

Addressing Modes

Bus System

Computer
Software And Hardware

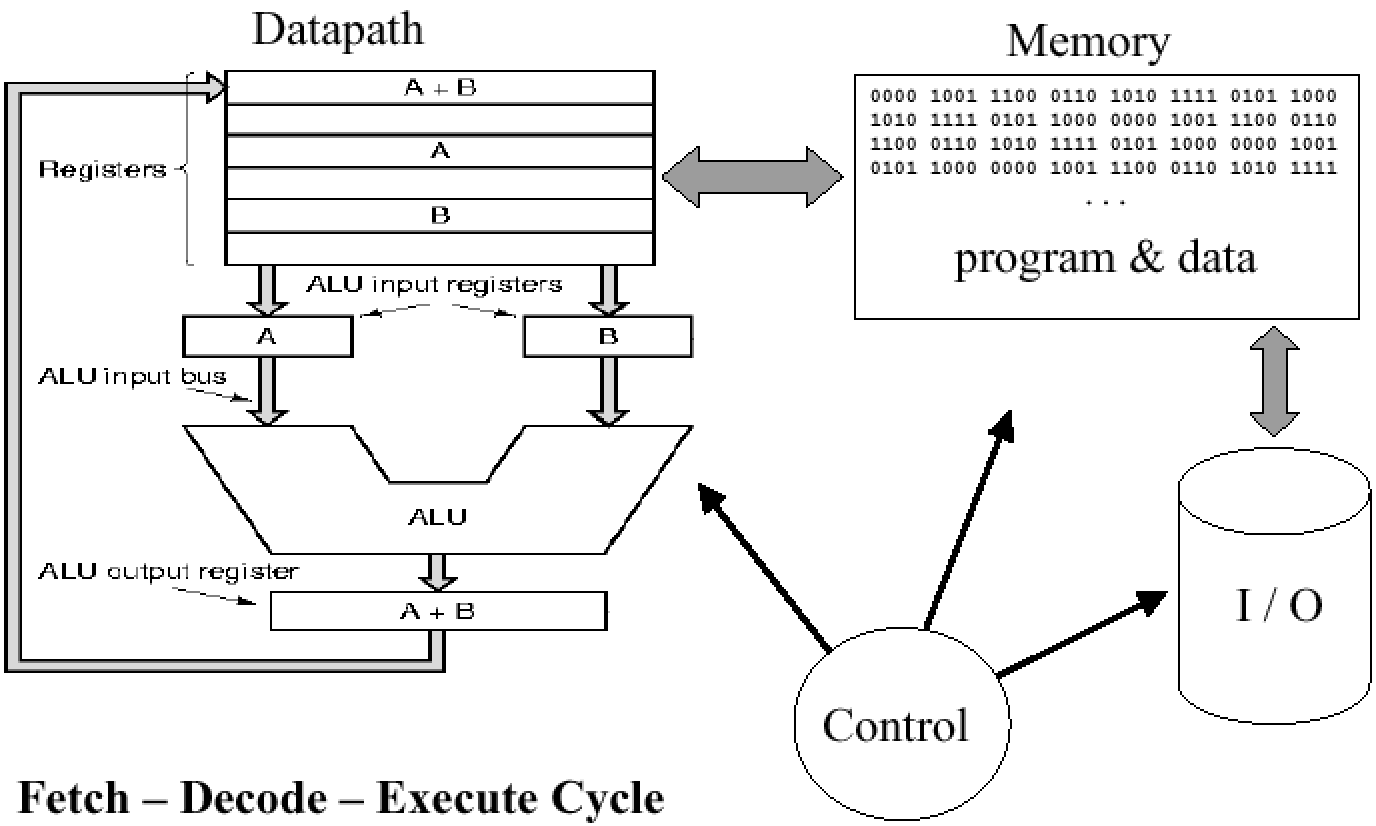
- A term that is often used interchangeably with computer architecture is Instruction Set Architecture (ISA).

- The ISA defines instruction formats, instruction opcodes, registers, instruction and data memory; the effect of executed instructions on the registers and memory; and an algorithm for controlling instruction execution.

- Computer organization refers to the operational units and their interconnections that realize the architectural specifications.

- Examples of architectural attributes include the instruction set, the number of bits used to represent various data types (e.g., numbers, characters), I/O mechanisms, and techniques for addressing memory.

Basic ISA cycle


Datapath
A + B
A
B
Registers
ALU input registers
A
B
ALU input bus
ALU
ALU output register
A + B
Memory
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
. . .
program & data
I / O
Control

Fetch – Decode – Execute Cycle

- Organizational attributes include those hardware details transparent to the programmer, such as control signals; interfaces between the computer and peripherals; and the memory technology used.
- For example,
  - it is an architectural design issue whether a computer will have a multiply instruction.
  - It is an organizational issue whether that instruction will be implemented by a special multiply unit or by a mechanism that makes repeated use of the add unit of the system.

- The organizational decision may be based on the anticipated frequency of use of the multiply instruction, the relative speed of the two approaches, and the cost and physical size of a special multiply unit.
- Historically, and still today, the distinction between architecture and organization has been an important one.
-

- Many computer manufacturers offer a family of computer models, all with the same architecture but with differences in organization.

- Consequently, the different models in the family have different price and performance characteristics.

- Furthermore, a particular architecture may span many years and encompass a number of different computer models, its organization changing with changing technology.

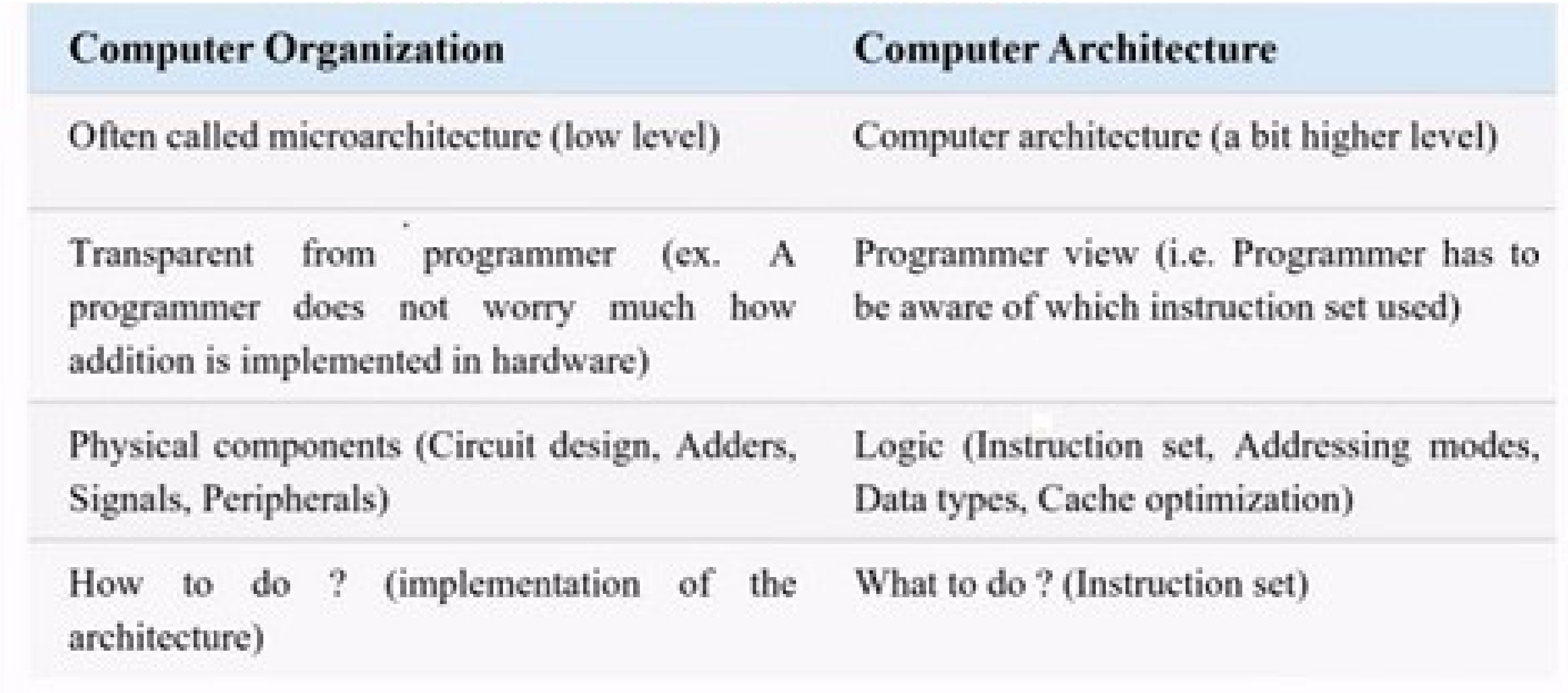
- A prominent example of both these phenomena is the IBM System/370 architecture.
  - This architecture was first introduced in 1970 and included a number of models.
  - The customer with modest requirements could buy a cheaper, slower model and, if demand increased, later upgrade to a more expensive, faster model without having to abandon software that had already been developed.

- Over the years, IBM has introduced many new models with improved technology to replace older models, offering the customer greater speed, lower cost, or both.

- These newer models retained the same architecture so that the customer’s software investment was protected.

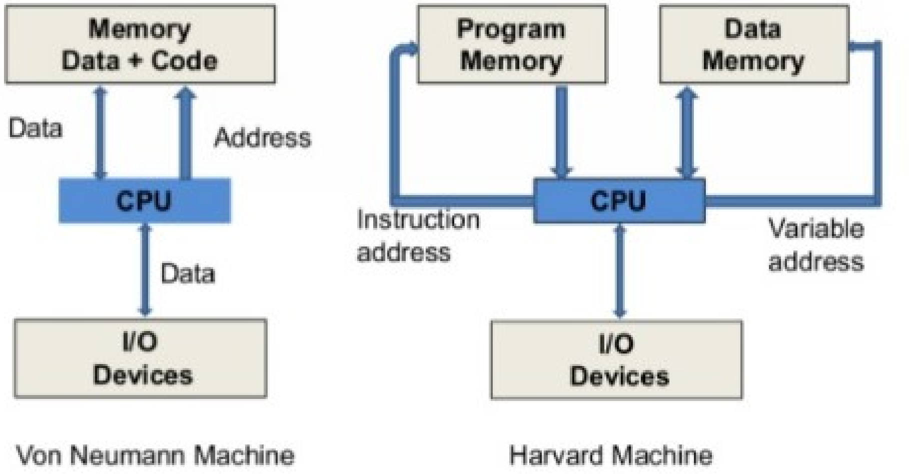
- Remarkably, the System/370 architecture, with a few enhancements, has survived to this day as the architecture of IBM’s mainframe product line.
- In a class of computers called microcomputers, the relationship between architecture and organization is very close.
- Changes in technology not only influence organization but also result in the introduction of more powerful and more complex architectures.
- Generally, there is less of a requirement for generation-to-generation compatibility for these smaller machines.
- Thus, there is more interplay between organizational and architectural design decisions.

Summary – Computer Organization vs Computer Architecture

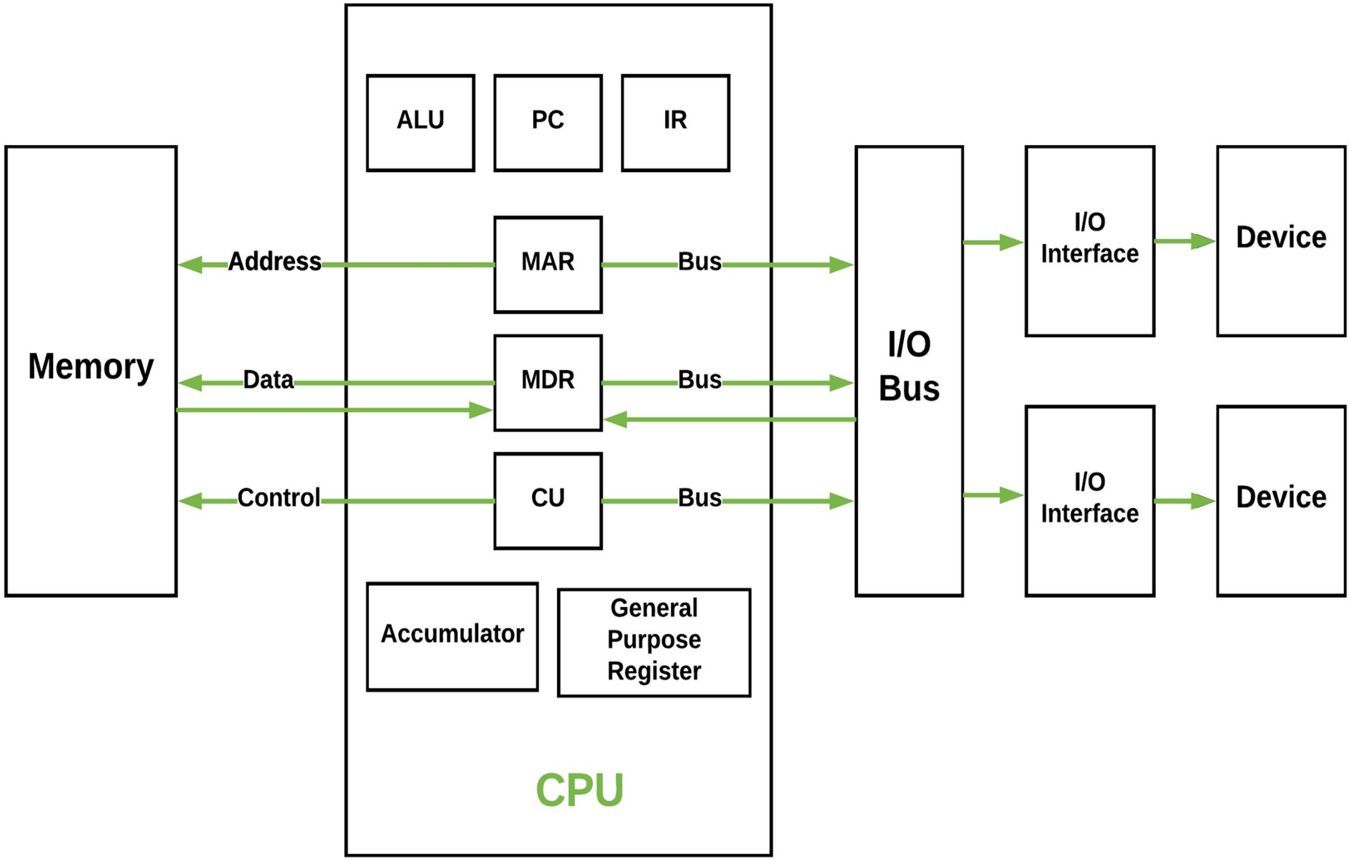


Computer Organization	Computer Architecture
Often called microarchitecture (low level)	Computer architecture (a bit higher level)
Transparent from programmer (ex. A programmer does not worry much how addition is implemented in hardware)	Programmer view (i.e. Programmer has to be aware of which instruction set used)
Physical components (Circuit design, Adders, Signals, Peripherals)	Logic (Instruction set, Addressing modes, Data types, Cache optimization)
How to do ? (implementation of the architecture)	What to do ? (Instruction set)

Von-Neumann Architecture vs Harvard Architecture


Memory
Data + Code
Data
Address
CPU
Data
I/O
Devices
Von Neumann Machine
Program
Memory
Data
Memory
CPU
Instruction
address
Variable
address
I/O
Devices
Harvard Machine

Von-Neumann Architecture vs Harvard Architecture


ALU
PC
IR
Memory
Address
MAR
Bus
Data
MDR
Bus
Control
CU
Bus
I/O Bus
I/O Interface
Device
I/O Interface
Device
Accumulator
General Purpose Register
CPU

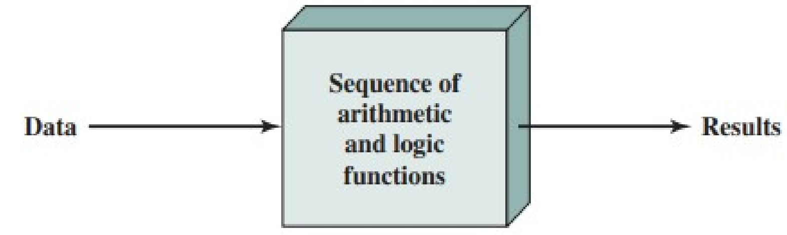
- Virtually all contemporary computer designs are based on concepts developed by John von Neumann at the Institute for Advanced Studies, Princeton. Such a design is referred to as the von Neumann architecture and is based on three key concepts:

  1. Data and instructions are stored in a single read–write memory.
  2. The contents of this memory are addressable by location, without regard to the type of data contained there.
  3. Execution occurs in a sequential fashion (unless explicitly modified) from one instruction to the next.

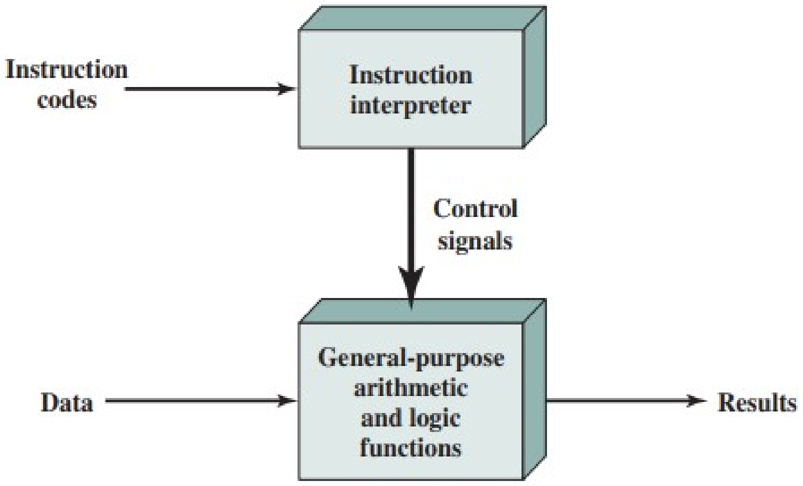
- There is a small set of basic logic components that can be combined in various ways to store binary data and perform arithmetic and logical operations on that data.

- If there is a particular computation to be performed, a configuration of logic components designed specifically for that computation could be constructed.
- We can think of the process of connecting the various components in the desired configuration as a form of programming.
- The resulting “program” is in the form of hardware and is termed a hardwired program.
- Now consider this alternative. Suppose we construct a general-purpose configuration of arithmetic and logic functions.
  - This set of hardware will perform various functions on data depending on control signals applied to the hardware.
  - In the original case of customized hardware, the system accepts data and produces results (Figure. a).

- With general-purpose hardware, the system accepts data and control signals and produces results.
- Thus, instead of rewiring the hardware for each new program, the programmer merely needs to supply a new set of control signals, and let us add to the general-purpose hardware a segment that can accept a code and generate control signals (Figure. (b)).


Data
Sequence of arithmetic and logic functions
Results

(a) Programming in hardware


Instruction codes
Instruction interpreter
Control signals
Data
General-purpose arithmetic and logic functions
Results

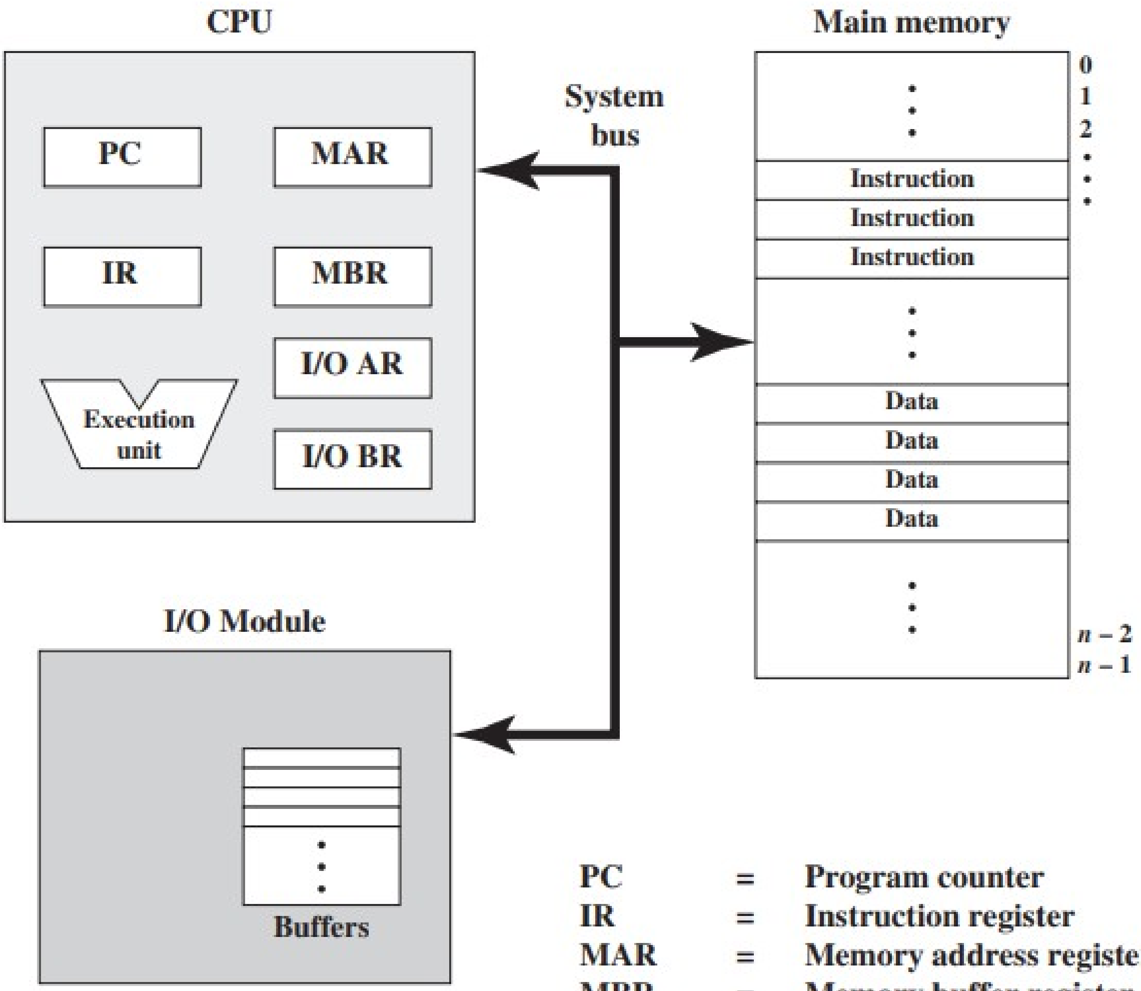
(b) Programming in software

- Programming is now much easier. Instead of rewiring the hardware for each new program, all we need to do is provide a new sequence of codes.
- Each code is, in effect, an instruction, and part of the hardware interprets each instruction and generates control signals.
- To distinguish this new method of programming, a sequence of codes or instructions is called software.
- Figure (b) indicates two major components of the system: an instruction interpreter and a module of general-purpose arithmetic and logic functions.
- These two constitute the CPU.
- Several other components are needed to yield a functioning computer.
- Data and instructions must be put into the system.

- For this we need some sort of input module.
- This module contains basic components for accepting data and instructions in some form and converting them into an internal form of signals usable by the system.
- A means of reporting results is needed, and this is in the form of an output module.
- Taken together, these are referred to as I/O components.
- One more component is needed.
- An input device will bring instructions and data in sequentially.
- But a program is not invariably executed sequentially; it may jump around (e.g., the IAS jump instruction).

- Similarly, operations on data may require access to more than just one element at a time in a predetermined sequence. Thus, there must be a place to temporarily store both instructions and data.
- That module is called memory, or main memory, to distinguish it from external storage or peripheral devices. Von Neumann pointed out that the same memory could be used to store both instructions and data.

- Figure illustrates these top-level components and suggests the interactions among them.
  - The CPU exchanges data with memory.
  - For this purpose, it typically makes use of two internal (to the CPU) registers:
- a memory address register (MAR), which specifies the address in memory for the next read or write, and a memory buffer register (MBR), which contains the data to be written into memory or receives the data read from memory.


CPU
PC
MAR
IR
MBR
I/O AR
I/O BR
Execution unit
System bus
Main memory
0
1
2
Instruction
Instruction
Instruction
Data
Data
Data
Data
n – 2
n – 1
I/O Module
Buffers

PC = Program counter
IR = Instruction register
MAR = Memory address register
MBR = Memory buffer register
I/O AR = Input/output address register
I/O BR = Input/output buffer register

- For this purpose, it typically makes use of two internal (to the CPU) registers:
- a memory address register (MAR), which specifies the address in memory for the next read or write, and a memory buffer register (MBR), which contains the data to be written into memory or receives the data read from memory. Similarly, an I/O address register (I/OAR) specifies a particular I/O device.
- An I/O buffer register (I/OBR) is used for the exchange of data between an I/O module and the CPU.
- A memory module consists of a set of locations, defined by sequentially numbered addresses. Each location contains a binary number that can be interpreted as either an instruction or data.
- An I/O module transfers data from external devices to CPU and memory, and vice versa.
- It contains internal buffers for temporarily holding these data until they can be sent on.

Features and Components of an ISA

- One of the most visible forms of evolution associated with computers is that of programming languages. As the cost of hardware has dropped, the relative cost of software has risen.

- Along with that, a chronic shortage of programmers has driven up software costs in absolute terms. Thus, the major cost in the life cycle of a system is software, not hardware.

Features and Components of an ISA

- Adding to the cost, and to the inconvenience, is the element of unreliability: it is common for programs, both system and application, to continue to exhibit new bugs after years of operation.
- The response from researchers and industry has been to develop ever more powerful and complex high- level programming languages.

- These high- level languages (HLLs):

(1) allow the programmer to express algorithms more concisely;

(2) allow the compiler to take care of details that are not important in the programmer’s expression of algorithms; and

(3) often support naturally the use of structured programming and/or object-oriented design.

- Alas, this solution gave rise to a perceived problem, known as the semantic gap, the difference between the operations provided in HLLs and those provided in computer architecture.

- Symptoms of this gap are alleged to include execution inefficiency, excessive machine program size, and compiler complexity.

- Designers responded with architectures intended to close this gap. Key features include large instruction sets, dozens of addressing modes, and various HLL statements implemented in hardware.

- An example of the latter is the CASE machine instruction on the VAX. Such complex instruction sets are intended to:
  - Ease the task of the compiler writer.
  - Improve execution efficiency, because complex sequences of operations can be implemented in microcode.

  - Provide support for even more complex and sophisticated HLLs.

- Meanwhile, a number of studies have been done over the years to determine the characteristics and patterns of execution of machine instructions generated from HLL programs.

- The results of these studies inspired some researchers to look for a different approach:
  - namely, to make the architecture that supports the HLL simpler, rather than more complex.

To understand the line of reasoning of the RISC advocates, we begin with a brief review of instruction execution characteristics.

- The aspects of computation of interest are as follows:
- **Operations performed:** These determine the functions to be performed by the processor and its interaction with memory.
- **Operands used:** The types of operands and the frequency of their use determine the memory organization for storing them and the addressing modes for accessing them.
- **Execution sequencing:** This determines the control and pipeline organization.

Instruction Execution Characteristics

- High Level Language (HLL) programs mostly comprises-
  - Assignment statements (e.g. A←B)
  - Conditional statements (e.g. IF, LOOP)
  - Procedure call/return (e.g. Functions in C)
- Key for ISA designer-
  - These statements should be supported in optimal fashion

RISC Architecture

- Earlier computers had small and simple instruction set. Why???
  - In the early 1980's, designers recommended that computers use fewer instructions with simple constructs so they can be executed much faster without accessing memory as often.

RISC Characteristics

- Relatively few instructions
  - 128 or less
- Relatively few addressing modes
  - Memory access is limited to LOAD and STORE instructions
- All operations done within the registers of the CPU
  - Use of overlapped register windows for optimization
- Fixed Length (4 Bytes), easily decoded instruction format
  - Instruction execution time consistent
- Hardwired control

RISC Processors

- MIPS R4000
  - First commercially available RISC processor
  - Supports thirty-two 64-bit registers
  - 128KB of high speed cache
- SPARC (Sun)
  - Based on Berkeley RISC model
- PowerPC (IBM)
- ARM processor family
- Apple iPods

CISC Architecture

- Later complexity and number of instructions both are increased. Why???
  - Simplify the compilation
- The trend into computer hardware complexity was influenced by various factors, such as
  - To provide support for more customer applications
  - Adding instructions that facilitate the translation from high level language into machine language programs
    - Single machine instruction for each high level language statement
  - Ex: VAX computer, IBM/370 computers, Intel x86 based processors, Motorola 68000 Series, Virtual address extension

CISC Characteristics

- A large number of instructions
- Some instructions for special tasks used infrequently
- A large variety of addressing modes (5 to 20)
- Variable length instruction formats
- Instructions that manipulate operands in memory
- However, it soon became apparent that a complex instruction set has a number of disadvantages
- These include a complex instruction decoding scheme, an increased size of the control unit,

RISC and CISC Comparison

- Computer performance equation
  - CPU Time = (instructions/program) x (avg. cycles/instruction) x (seconds/cycle)
- Example CISC Program
  - mov ax, 20
  - mov bx, 5
  - mul ax, bx

Example RISC Program

```
      mov ax,0
      mov bx, 20
      mov cx,5
again add ax, bx
      loop again
```

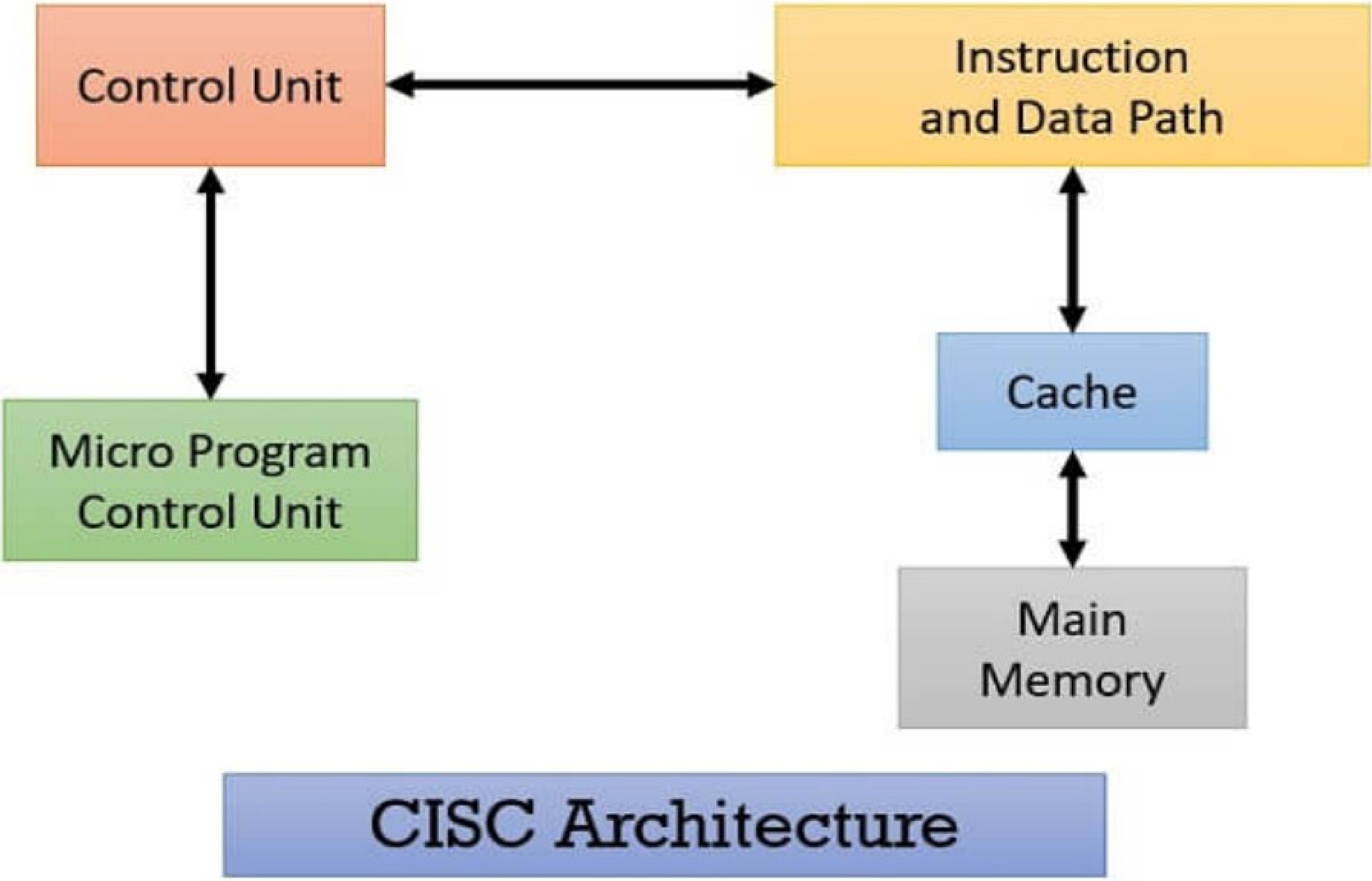
RISC vs CISC Performance Summary

- The CISC approach attempts to minimize the number of instructions per program by sacrificing the number of cycles per instruction.

- RISC does the opposite, reducing the cycles per instruction at the cost of the number of instructions per program.

RISC Vs. CISC Controversy

- Now the general trend in computer architecture and organization has been toward-
  - Increasing processor complexity
  - More instructions
  - More addressing modes
  - More registers
- No definitive test set of programs exists to compare both architectures
  - Performance varies with the programs
- Most commercially available machines are mixture of RISC and CISC characteristics



Control Unit

Instruction and Data Path

Micro Program Control Unit

Cache

Main Memory

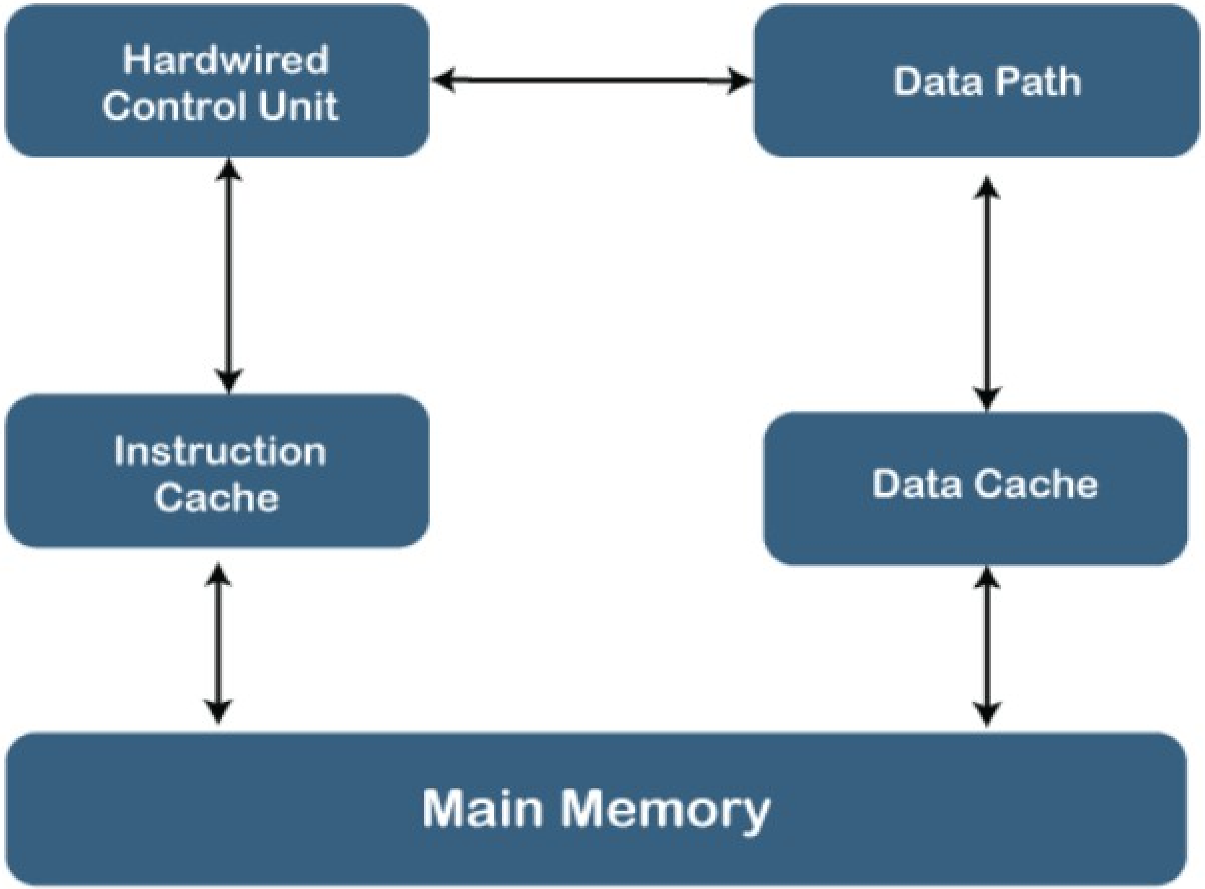
CISC Architecture

Advantages of CISC Processors

1. The code size is comparatively shorter which minimizes the memory requirement.
2. Execution of a single instruction accomplishes several low-level tasks.
3. Complex addressing mode makes the memory access flexible.
4. CISC instruction can directly access memory locations.

Disadvantages of CISC Processors

1. Though the code size is minimized but it requires several clock cycles to execute a single instruction. Thereby reduce the overall performance of the computer.
2. Implementing pipelining for CISC instruction is a bit complicated.
3. The hardware structure needs to be more complex to simplify software implementation.
4. Designed to minimize the memory requirement when memory was smaller and costlier. But situation is different today


Hardwired Control Unit
Data Path
Instruction Cache
Data Cache
Main Memory

RISC Architecture

Advantages of RISC Processors

1. The RISC processor's performance is better due to the simple and limited number of the instruction set.

2. It requires several transistors that make it cheaper to design.

3. RISC allows the instruction to use free space on a microprocessor because of its simplicity.

4. RISC processor is simpler than a CISC processor because of its simple and quick design, and it can complete its work in one clock cycle.

Disadvantages of RISC Processors

1. The RISC processor's performance may vary according to the code executed because subsequent instructions may depend on the previous instruction for their execution in a cycle.

2. Programmers and compilers often use complex instructions.

3. RISC processors require very fast memory to save various instructions that require a large collection of cache memory to respond to the instruction in a short time.

CISC vs RISC

- After the initial enthusiasm for RISC machines, there has been a growing realization that

  (1) RISC designs may benefit from the inclusion of some CISC features and that

  (2) CISC designs may benefit from the inclusion of some RISC features.

- The result is that the more recent RISC designs, notably the PowerPC, are no longer “pure” RISC and the more recent CISC designs, notably the Pentium II and later Pentium models, do incorporate some RISC characteristics.

CISC vs RISC

- For many years, the general trend in computer architecture and organization has been toward increasing processor complexity:
  - more instructions, more addressing modes, more specialized registers, and so on.
  - The RISC movement represents a fundamental break with the philosophy behind that trend.

- Naturally, the appearance of RISC systems, and the publication of papers by its proponents extolling RISC virtues, led to a reaction from those involved in the design of CISC architectures.

- The work that has been done on assessing merits of the RISC approach can be grouped into two categories:

  - **Quantitative:** Attempts to compare program size and execution speed of programs on RISC and CISC machines that use comparable technology.

  - **Qualitative:** Examines issues such as high- level language support and optimum use of VLSI real estate.

- Most of the work on quantitative assessment has been done by those working on RISC systems, and it has been, by and large, favorable to the RISC approach.
- Others have examined the issue and come away unconvinced.

There are several problems with attempting such comparisons:

- There is no pair of RISC and CISC machines that are comparable in life-cycle cost, level of technology, gate complexity, sophistication of compiler, operating system support, and so on.
- No definitive test set of programs exists. Performance varies with the program.

- Most of the comparative analysis on RISC has been done on “toy” machines rather than commercial products.

- Furthermore, most commercially available machines advertised as RISC possess a mixture of RISC and CISC characteristics.

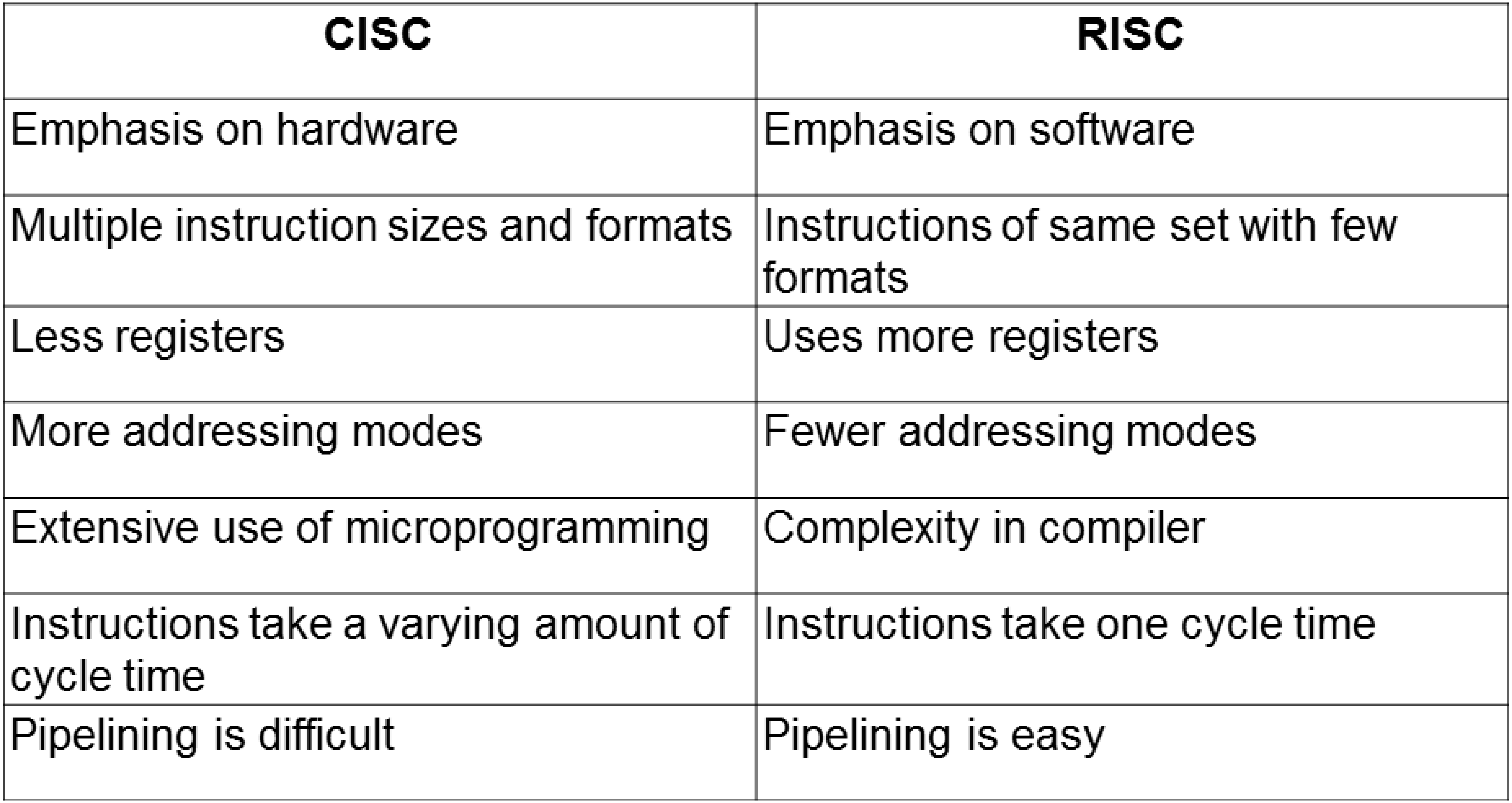
- Thus, a fair comparison with a commercial, “pure-play” CISC machine (e.g., VAX, Pentium) is difficult.

- The qualitative assessment is, almost by definition, subjective.
  - Several researchers have turned their attention to such an assessment, but the results are, at best, ambiguous, and certainly subject to rebuttal and, of course, counter-rebuttal [COLW85b].
  - In more recent years, the RISC versus CISC controversy has died down to a great extent. This is because there has been a gradual convergence of the technologies.

- As chip densities and raw hardware speeds increase, RISC systems have become more complex.

- At the same time, in an effort to squeeze out maximum performance, CISC designs have focused on issues traditionally associated with RISC, such as an increased number of general-purpose registers and increased emphasis on instruction pipeline design.

RISC vs. CISC



CISC	RISC
Emphasis on hardware	Emphasis on software
Multiple instruction sizes and formats	Instructions of same set with few formats
Less registers	Uses more registers
More addressing modes	Fewer addressing modes
Extensive use of microprogramming	Complexity in compiler
Instructions take a varying amount of cycle time	Instructions take one cycle time
Pipelining is difficult	Pipelining is easy

The CISCs: x86, VAX (Virtual Address eXtension to PDP-11)

- Variable length instructions: 1-321 bytes!!!
- 14 GPRs + PC + stack-pointer + condition codes
- Data sizes: 8, 16, 32, 64, 128 bit, decimal, string
- Memory-memory instructions for all data sizes

The RISCs: MIPS, PA-RISC, SPARC, PowerPC, Alpha, ARM

- 32-bit instructions
- 32 integer registers, 32 floating point registers, load-store
- 64-bit virtual address space
- Few addressing modes (Alpha has one, SPARC/PowerPC have more)

Design Goals

- The ultimate goals of the ISA designer are:
  - To create an ISA that allows for fast hardware implementations
  - To simplify choices for the compiler
  - To ensure the longevity of the ISA by anticipating future technology trends
- Often trade-offs (particularly between 1 & 2)
- Examples ISAs: X86, PowerPC, SPARC, ARM, MIPS, IA-64
- May have multiple hardware implementations of the same ISA
- Example: i386, i486, Pentium, Pentium Pro, Pentium II, Pentium III, Pentium IV

THANK YOU

BITS Pilani
Pilani Campus

BITS Pilani
Pilani Campus

COMPUTING AND DESIGN SESSION 4

Course design by - Dr. Lucy J. Gudino &
Dr Chandrasekhar
Faculty – Thangakumar J
WILP & Department of CS & IS

Last Session

List of Topic Title	Text/Ref Book/external resource
• Computer Architecture Vs. Organization (Abstraction vs. Realization) • Von-Neumann Architecture vs Harvard Architecture • Features and Components of an ISA. • Design Goals	R6 (1.1) R6 (3.1) R6 (15.1)

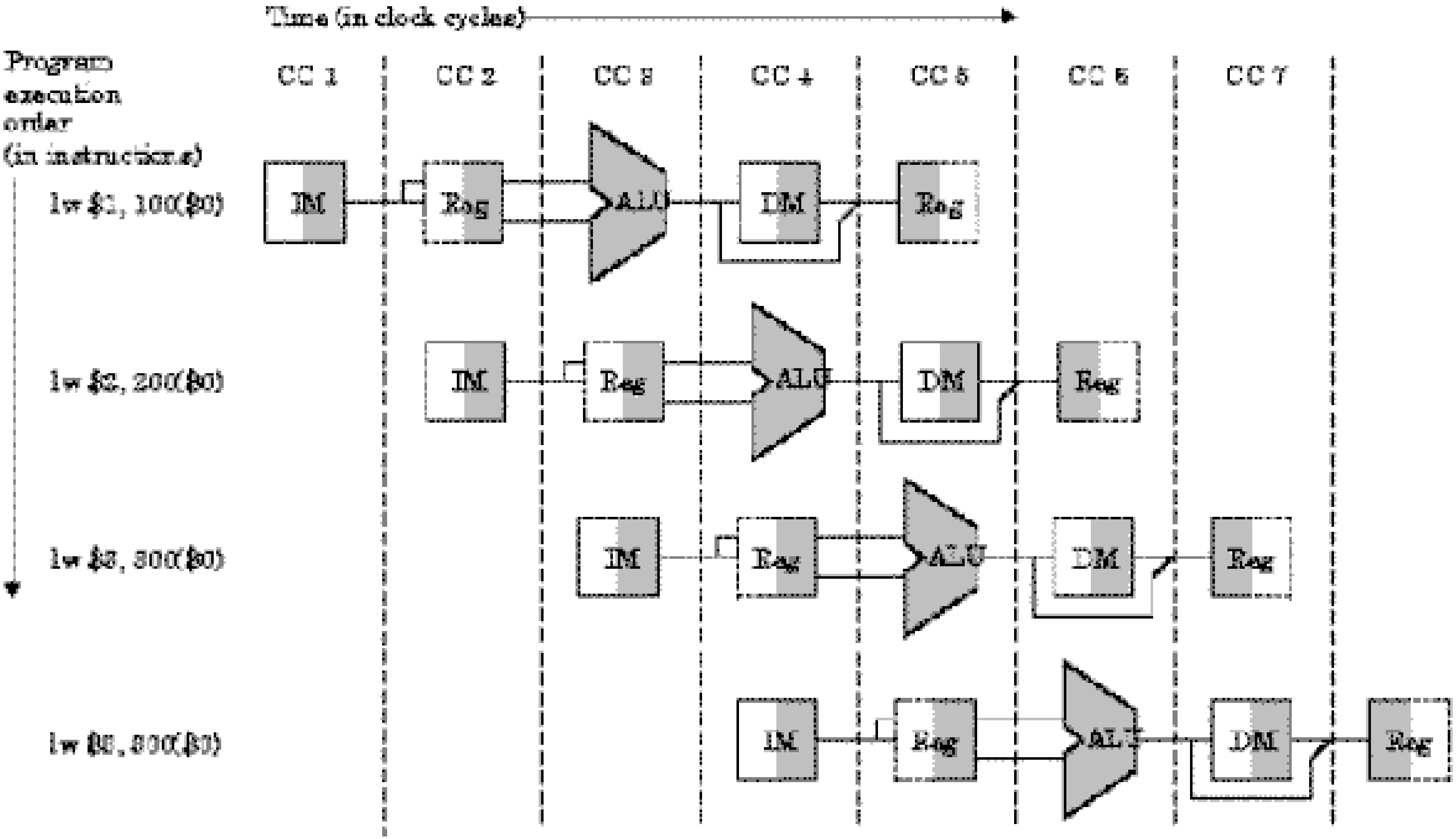
ISA as the Interface

Today's topic

- Understanding MIPS/RISC architecture

MIPS Architecture

- MIPS follows RISC principles and based on Harvard Architecture
- MIPS architecture is a register architecture
- MIPS ISA is Load/Store architecture
- R2000 is a 32-bit processor
- Uses fixed length instruction format.


Time (in clock cycles)
Program
execution
order
(in instructions)
CC 1
CC 2
CC 3
CC 4
CC 5
CC 6
CC 7
lw $1, 100($0)
lw $2, 200($0)
lw $3, 300($0)
lw $3, 300($0)
IM
Reg
ALU
DM
Reg

R2000

- The R2000 is a 32-bit microprocessor chip set developed by MIPS Computer Systems that implemented the MIPS I instruction set architecture (ISA).
- Introduced in January 1986, it was the first commercial implementation of the MIPS architecture and the first commercial RISC processor available to all companies.
- The R2000 competed with Digital Equipment Corporation (DEC) VAX minicomputers and with Motorola 68000 and Intel Corporation 80386 microprocessors.
- R2000 users included Ardent Computer, DEC, Silicon Graphics, Northern Telecom and MIPS's own Unix

MIPS Register Set

Types of Registers

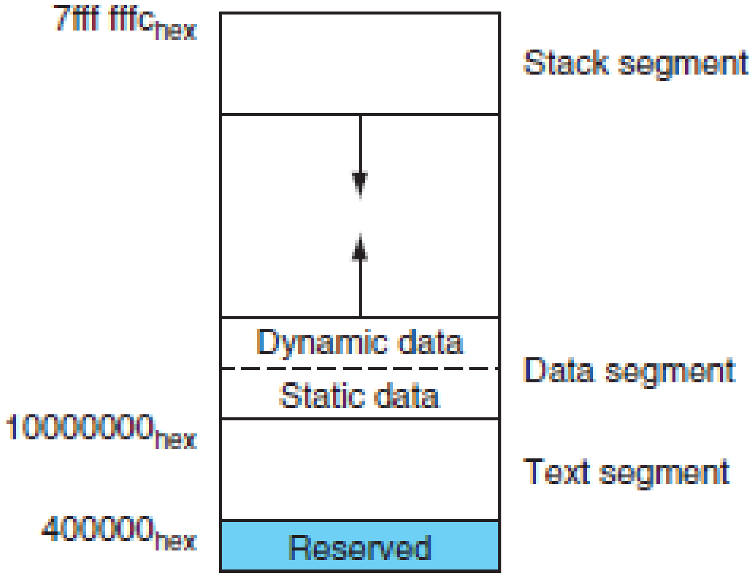
- 32 general-purpose registers ($0 – $31)
- Program Counter (PC)
- Two special Purpose register (HI and LO)
  - Used to hold the results of integer *multiply* and *divide* instruction.
  - integer *multiply* operation, HI and LO register hold the 64-bit result.
  - integer *divide* operation, the 32-bit quotient is stored in the LO and the remainder in the HI register.

General-Purpose Register Usage Convention

Register name	Number	Intended usage
Zero	0	Constant 0
at	1	Reserved for assembler
v0,v1	2,3	Values for function results and Exp evaluation
a0,a1,a2,a3	4-7	Arguments 1-4
t0-t7	8-15	Temporary (not preserved across call)
s0-s7	16-23	Saved temporary(preserved across call)
t8,t9	24,25	Temporary (not preserved across call)
k0,k1	26,27	Reserved for OS kernel
gp	28	Pointer to Global area
sp	29	Stack Pointer
fp	30	Frame pointer(if needed);Otherwise, a saved register $s8
ra	31	Return address(used by a procedure call)

Memory Usage

- A program's address space consists of three parts:
  - Code/Text segment
  - Data segment
  - Stack segment


7fff fffc$_{hex}$
Stack segment
Dynamic data
Static data
Data segment
10000000$_{hex}$
Text segment
400000$_{hex}$
Reserved

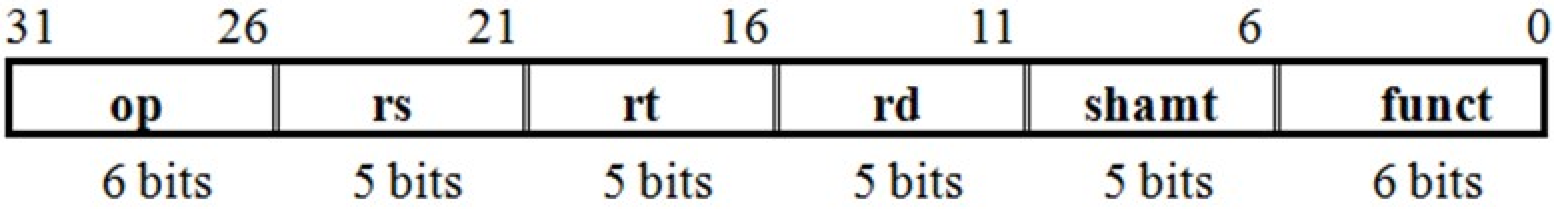
Instruction Format

- Fixed-length instruction format
- 32-bits long
- Three different instruction formats:

  1. Immediate (I-type)
  2. Jump (J-type)
  3. Register (R-type)
- Meaning of various fields in the instruction format
  - op: Opcode
  - rs: The first register source operand
  - rt: The second register source operand
  - rd: The register destination operand
  - shamt / sa: shift amount
  - funct: function code

Instruction Format

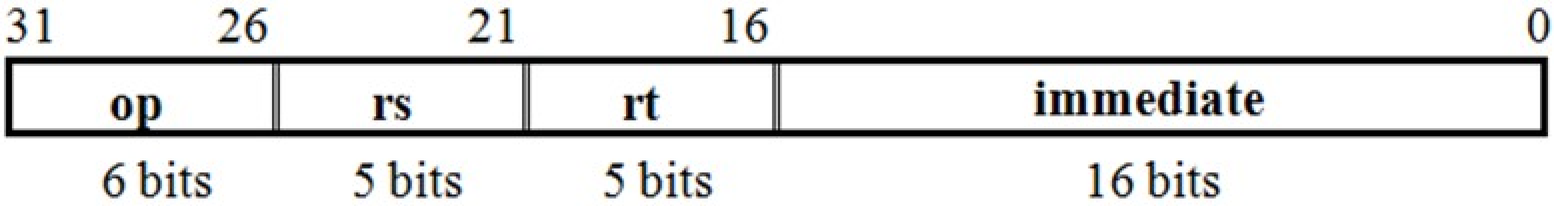
R-Type:



31 26 21 16 11 6 0

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

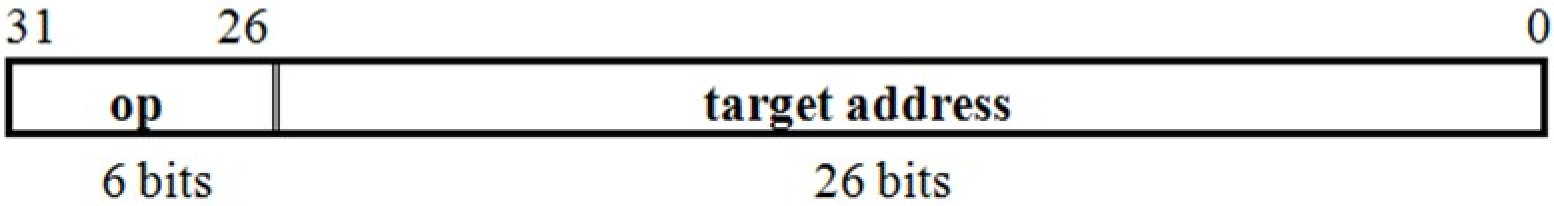
I-Type:



31 26 21 16 0

op	rs	rt	immediate
6 bits	5 bits	5 bits	16 bits

J-Type:

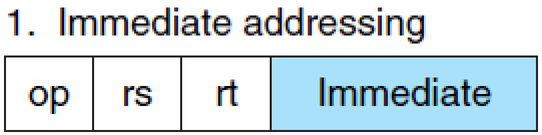


31 26 0

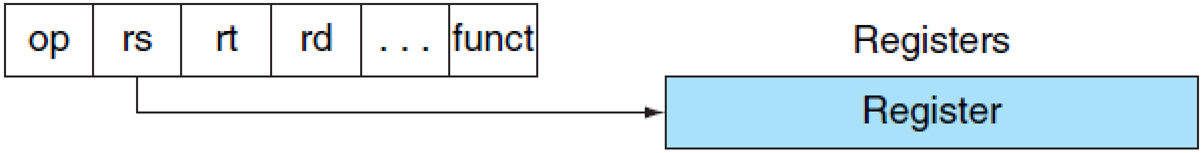
op	target address
6 bits	26 bits

Addressing modes

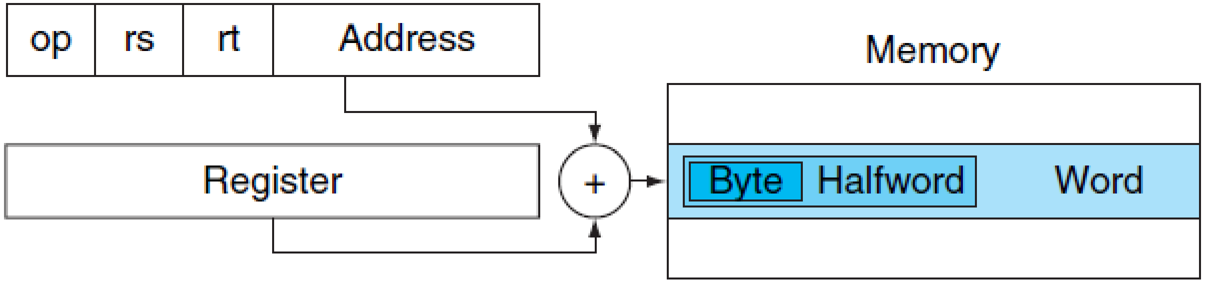
- Immediate Addressing Mode


1. Immediate addressing
op
rs
rt
Immediate

- Register Addressing Mode

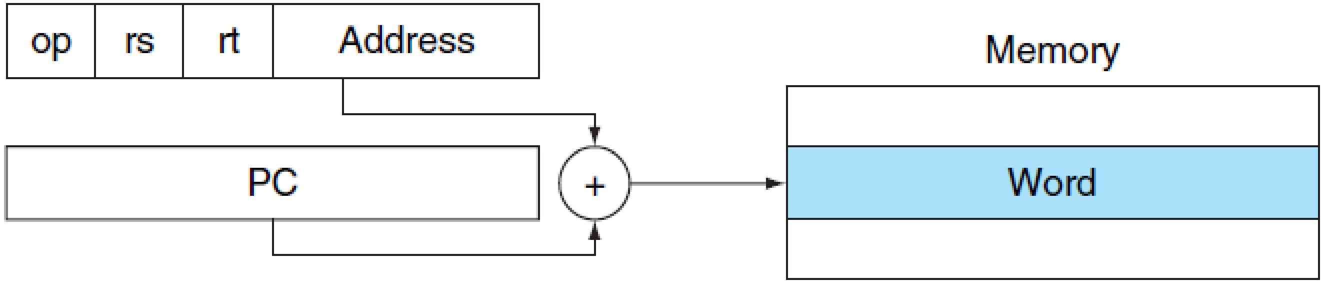

op
rs
rt
rd
. . .
funct
Registers
Register

- Base or Displacement Addressing Mode

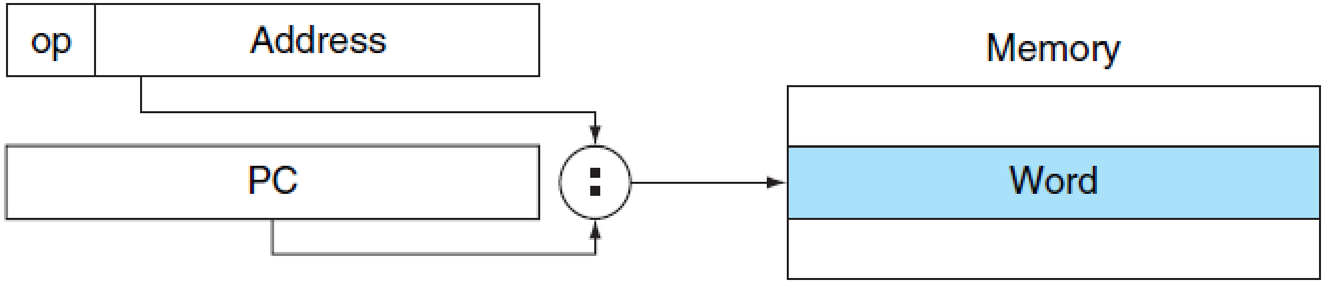

op
rs
rt
Address
Memory
Register
+
Byte
Halfword
Word

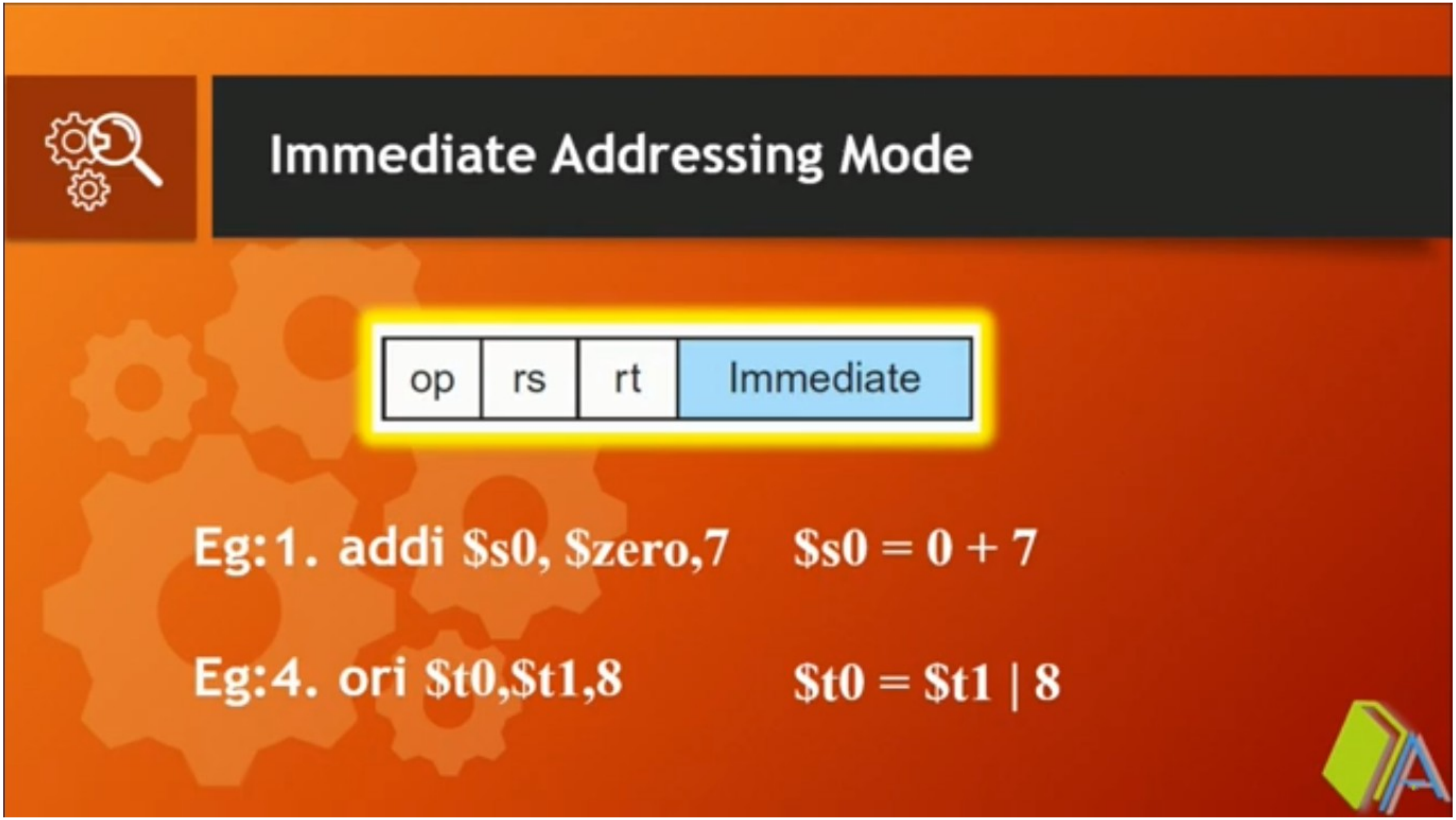
Addressing modes

- PC- Relative Addressing Mode


op
rs
rt
Address
Memory
PC
+
Word

- Pseudo Direct Addressing Mode


op
Address
Memory
PC
:
Word

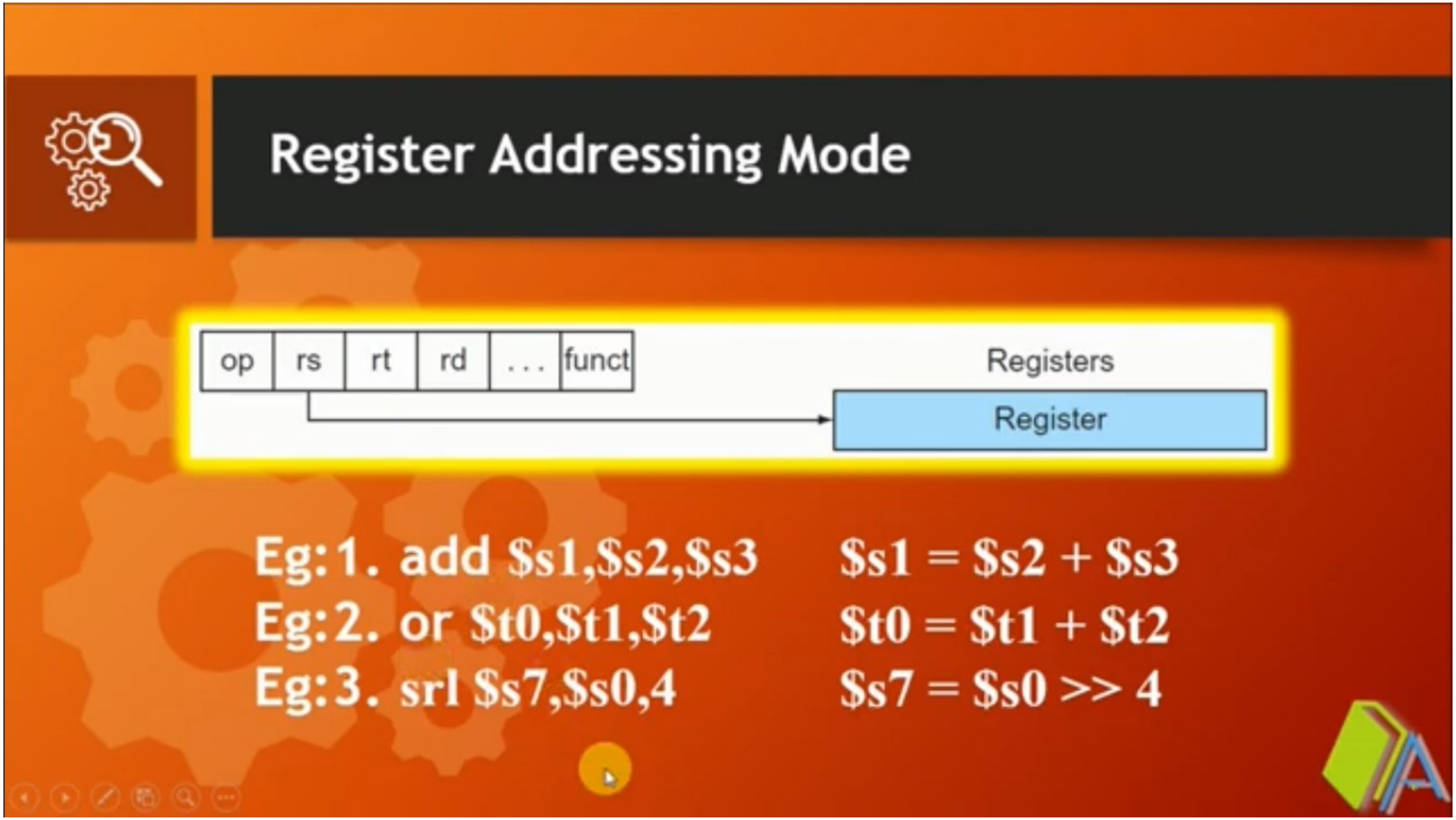


Immediate Addressing Mode

op	rs	rt	Immediate

Eg:1. addi $s0, $zero,7 $s0 = 0 + 7

Eg:4. ori $t0,$t1,8 $t0 = $t1 | 8



Register Addressing Mode

op	rs	rt	rd	. . .	funct

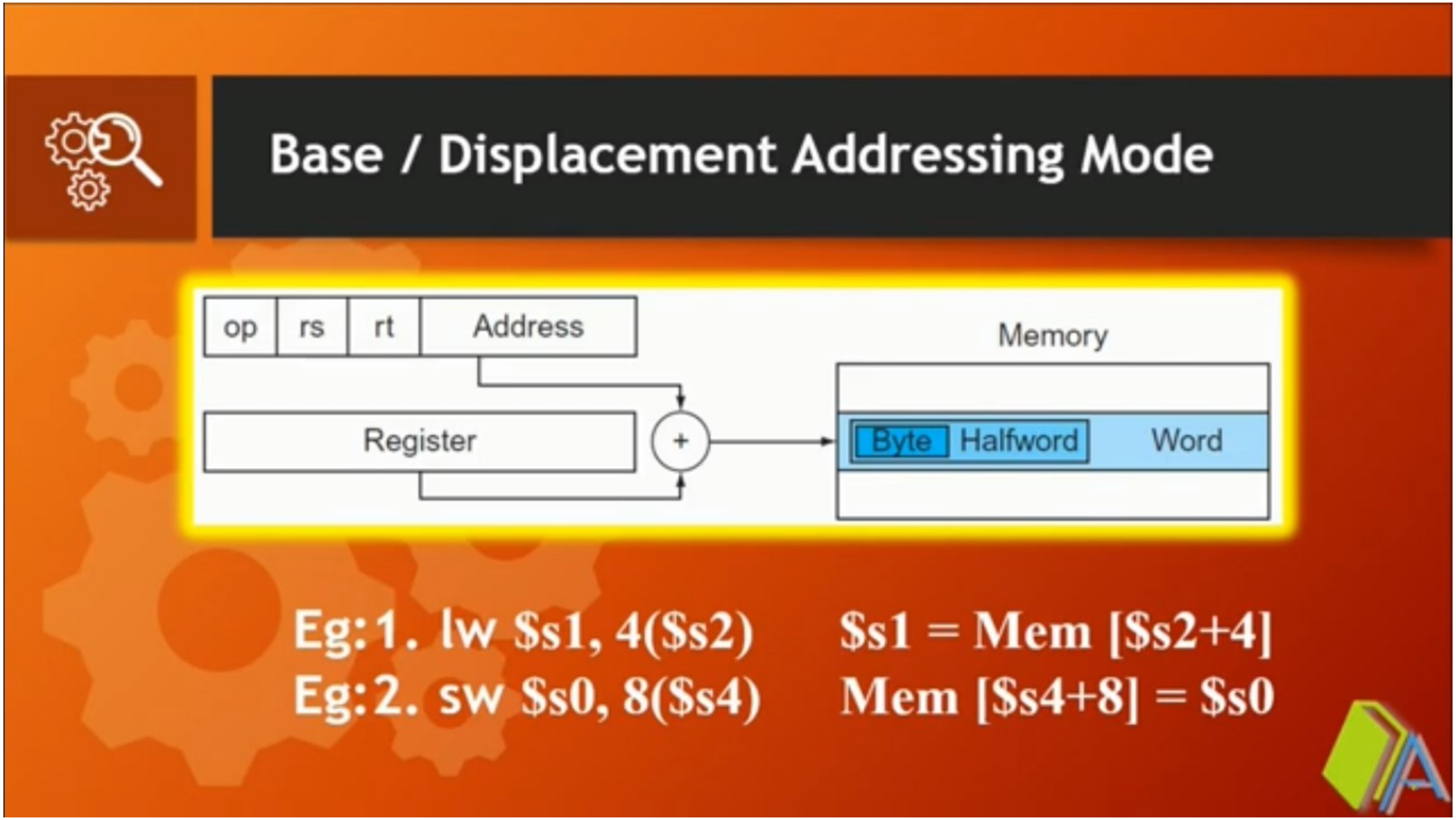
Registers

Register

Eg:1. add $s1,$s2,$s3 $s1 = $s2 + $s3

Eg:2. or $t0,$t1,$t2 $t0 = $t1 + $t2

Eg:3. srl $s7,$s0,4 $s7 = $s0 >> 4

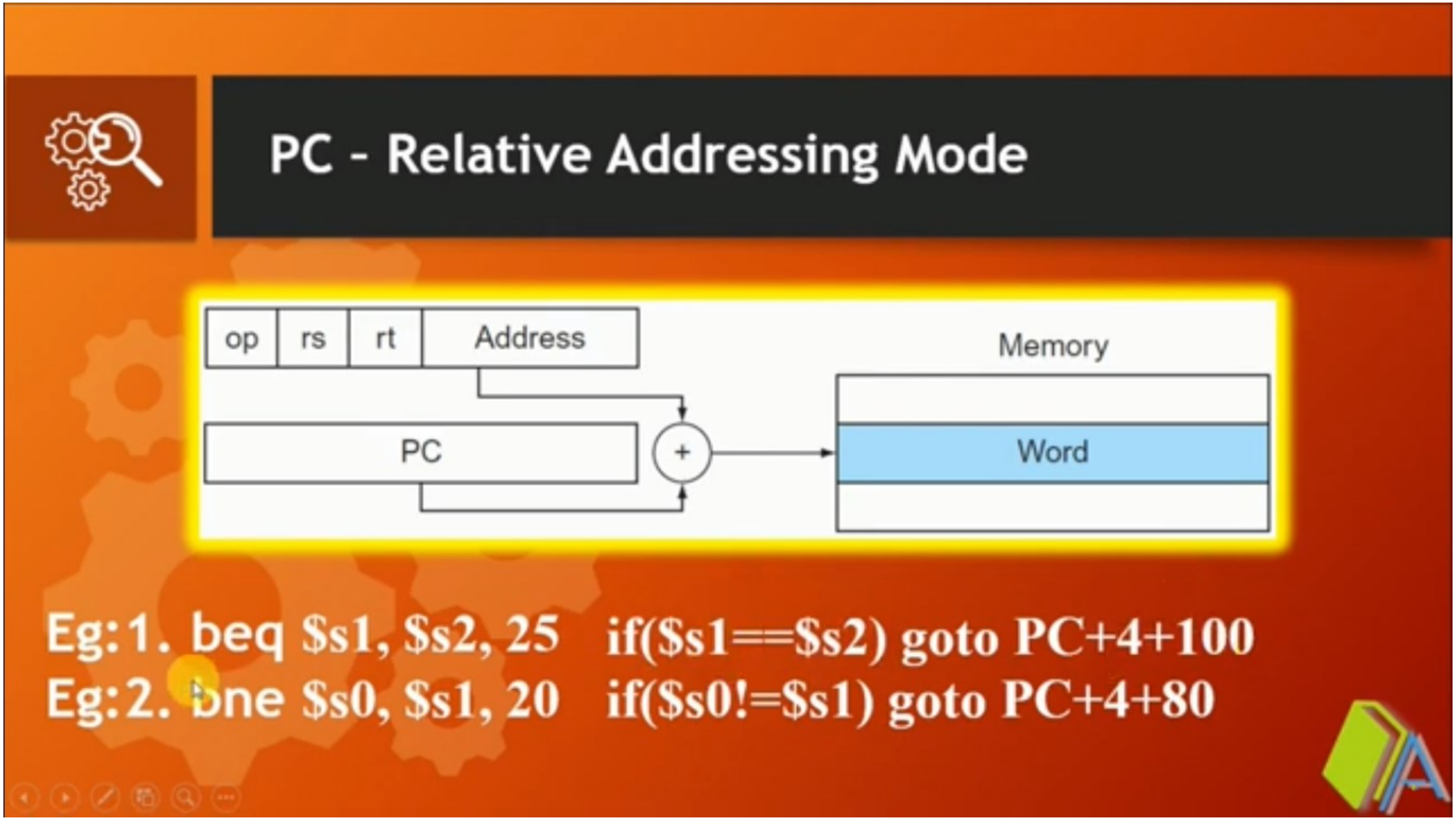


Base / Displacement Addressing Mode

op
rs
rt
Address
Register
+
Memory
Byte
Halfword
Word

Eg:1. lw $s1, 4($s2) $s1 = Mem [$s2+4]

Eg:2. sw $s0, 8($s4) Mem [$s4+8] = $s0

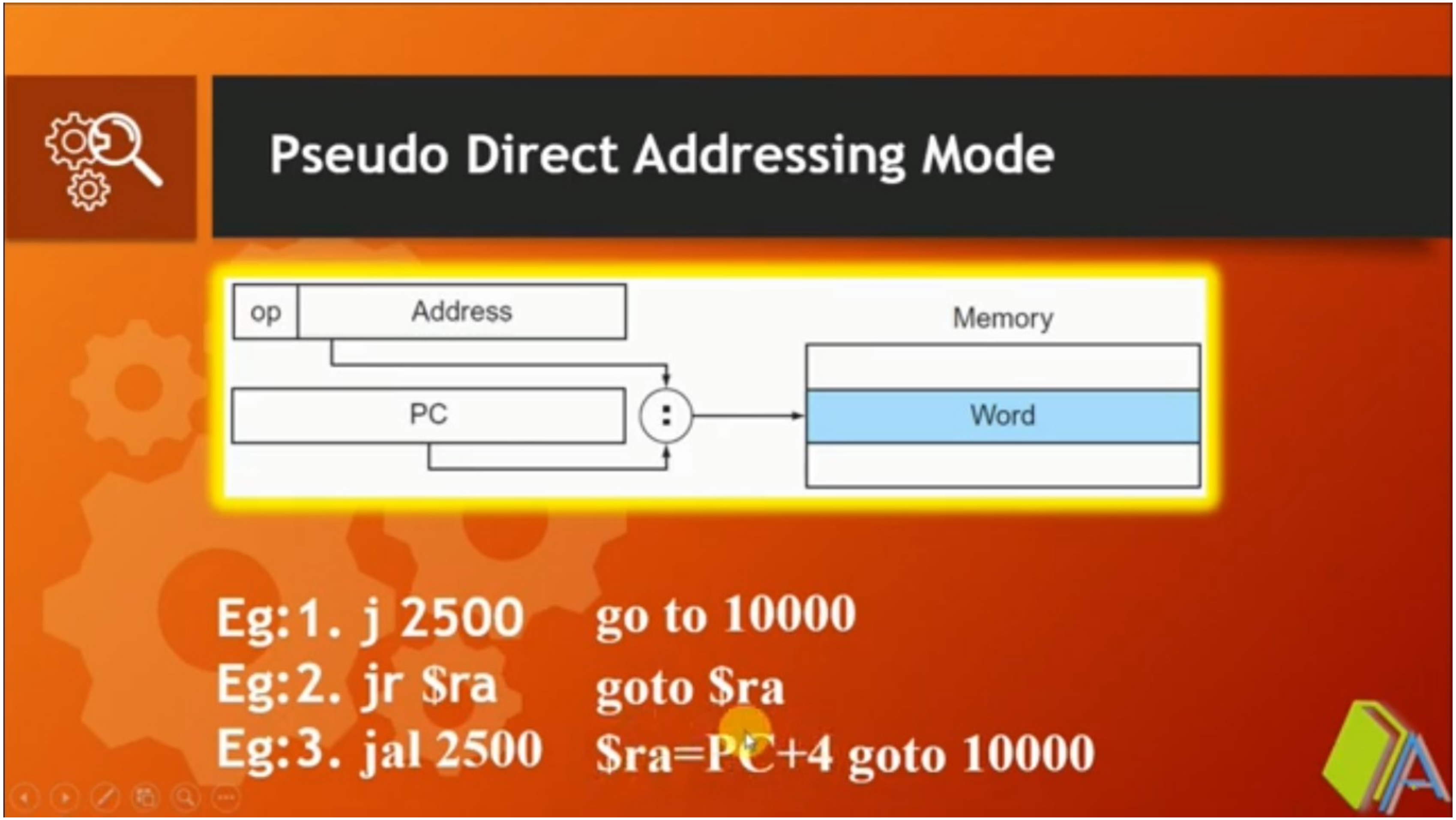


PC - Relative Addressing Mode

op
rs
rt
Address
Memory
PC
+
Word

Eg:1. beq $s1, $s2, 25 if($s1==$s2) goto PC+4+100

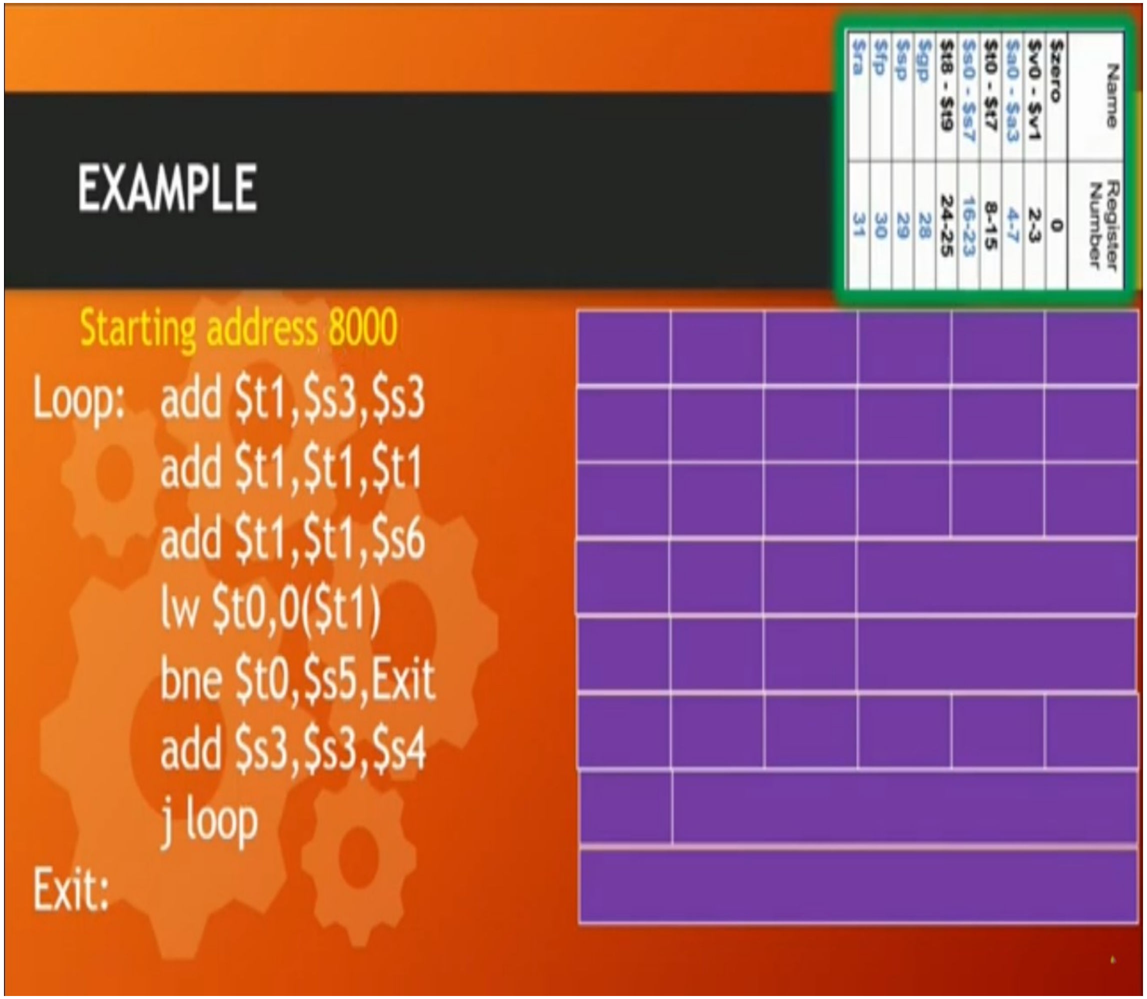
Eg:2. bne $s0, $s1, 20 if($s0!=$s1) goto PC+4+80



Pseudo Direct Addressing Mode

op
Address
PC
Memory
Word

Eg:1. j 2500 go to 10000
Eg:2. jr $ra goto $ra
Eg:3. jal 2500 $ra=PC+4 goto 10000



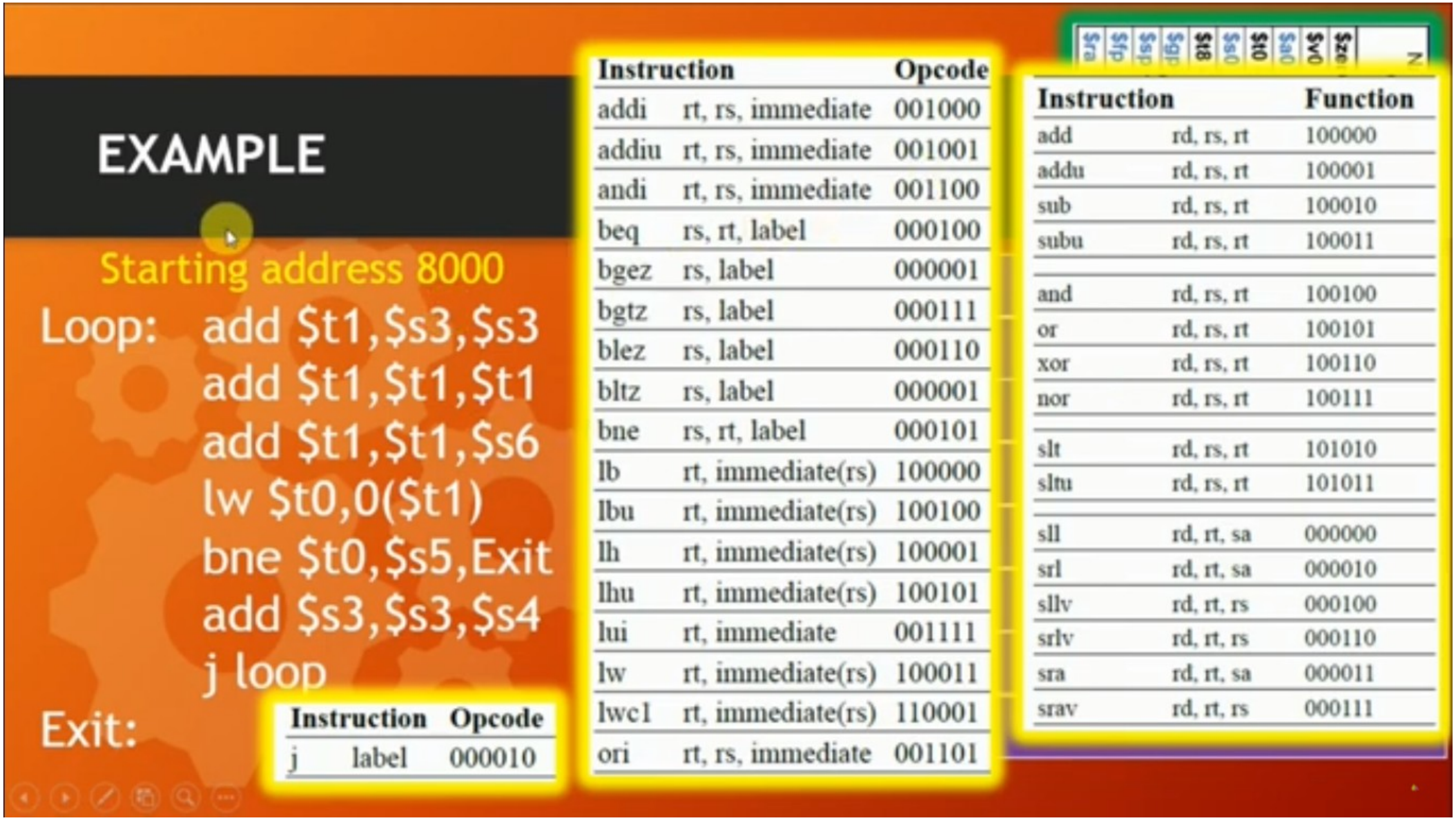
EXAMPLE

Name	Register Number
$zero	0
$v0 - $v1	2-3
$a0 - $a3	4-7
$t0 - $t7	8-15
$s0 - $s7	16-23
$t8 - $t9	24-25
$gp	28
$sp	29
$fp	30
$ra	31

Starting address 8000

```
Loop:   add $t1,$s3,$s3
        add $t1,$t1,$t1
        add $t1,$t1,$s6
        lw $t0,0($t1)
        bne $t0,$s5,Exit
        add $s3,$s3,$s4
        j loop
Exit:
```

Register name	Number
Zero	0
at	1
v0,v1	2,3
a0,a1,a2,a3	4-7
t0-t7	8-15
s0-s7	16-23
t8,t9	24,25
k0,k1	26,27
gp	28
sp	29
fp	30
ra	31



EXAMPLE

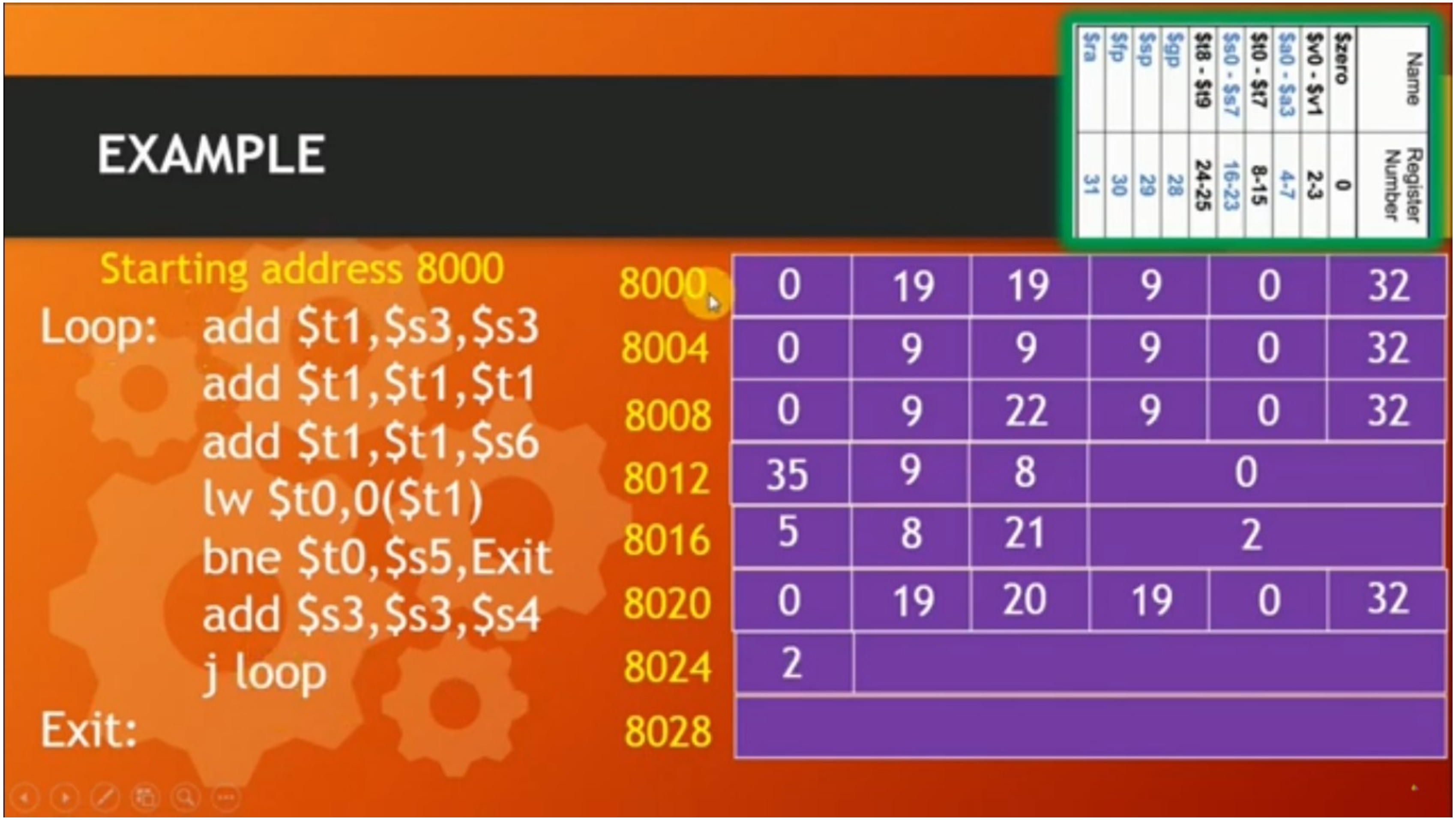
Starting address 8000

```
Loop:   add $t1,$s3,$s3
        add $t1,$t1,$t1
        add $t1,$t1,$s6
        lw $t0,0($t1)
        bne $t0,$s5,Exit
        add $s3,$s3,$s4
        j loop
Exit:
```

Instruction		Opcode
j	label	000010

Instruction		Opcode
addi	rt, rs, immediate	001000
addiu	rt, rs, immediate	001001
andi	rt, rs, immediate	001100
beq	rs, rt, label	000100
bgez	rs, label	000001
bgtz	rs, label	000111
blez	rs, label	000110
bltz	rs, label	000001
bne	rs, rt, label	000101
lb	rt, immediate(rs)	100000
lbu	rt, immediate(rs)	100100
lh	rt, immediate(rs)	100001
lhu	rt, immediate(rs)	100101
lui	rt, immediate	001111
lw	rt, immediate(rs)	100011
lwc1	rt, immediate(rs)	110001
ori	rt, rs, immediate	001101

Instruction		Function
add	rd, rs, rt	100000
addu	rd, rs, rt	100001
sub	rd, rs, rt	100010
subu	rd, rs, rt	100011
and	rd, rs, rt	100100
or	rd, rs, rt	100101
xor	rd, rs, rt	100110
nor	rd, rs, rt	100111
slt	rd, rs, rt	101010
sltu	rd, rs, rt	101011
sll	rd, rt, sa	000000
srl	rd, rt, sa	000010
sllv	rd, rt, rs	000100
srlv	rd, rt, rs	000110
sra	rd, rt, sa	000011
srav	rd, rt, rs	000111



EXAMPLE

Name	Register Number
$zero	0
$v0 - $v1	2-3
$a0 - $a3	4-7
$t0 - $t7	8-15
$s0 - $s7	16-23
$t8 - $t9	24-25
$gp	28
$sp	29
$fp	30
$ra	31

Starting address 8000

```
Loop:   add $t1,$s3,$s3
        add $t1,$t1,$t1
        add $t1,$t1,$s6
        lw $t0,0($t1)
        bne $t0,$s5,Exit
        add $s3,$s3,$s4
        j loop
Exit:
```

8000	0	19	19	9	0	32
8004	0	9	9	9	0	32
8008	0	9	22	9	0	32
8012	35	9	8	0		
8016	5	8	21	2		
8020	0	19	20	19	0	32
8024	2					
8028						

MIPS Instruction Set

- Arithmetic Instructions (ADD, SUB etc.)
- Logical Instructions (OR, AND, SHIFT, etc.)
- Data Transfer Instructions (Load and Store)
- Decision Making Instructions (J, BNE, BEQ, etc.)
- Stack Related Instructions (PUSH and POP)

A Basic MIPS Implementation

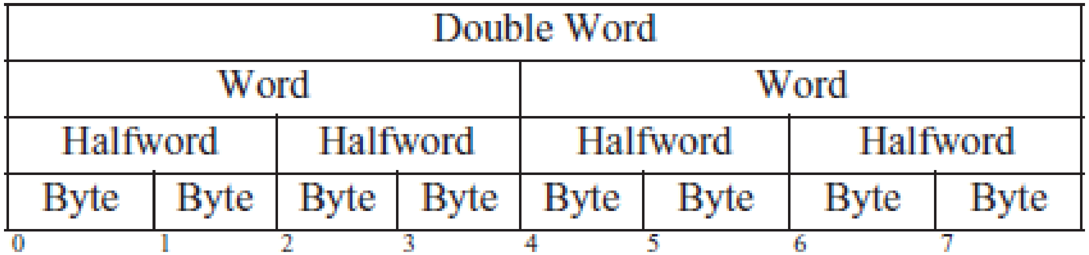
- Basic implementation includes a subset of core MIPS instruction set
  - Memory reference instruction
    - lw and sw
  - Arithmetic and logical instructions
    - add, sub, and, or and slt (set if less than)
  - Branch instructions
    - beq and j

Memory Reference Instruction - lw

- copies data from memory to register
- Format : lw reg, address

http://alumni.cs.ucr.edu/~vladimir/cs161/mips.html

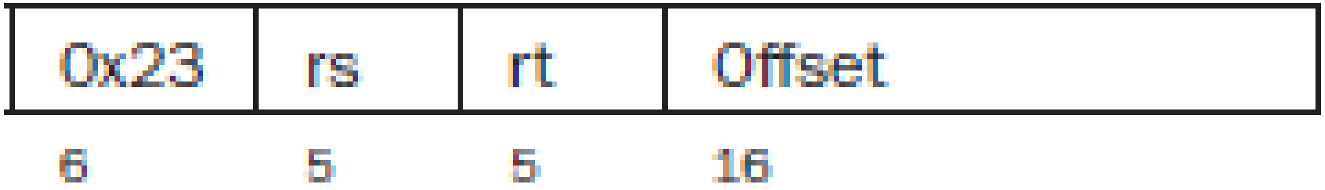
- Example : lw $s1, 100($s2)
- $s1 ← memory[100 +$s2]
- Alignment restriction
- MIPS is Big endian
- Spilling registers



Double Word							
Word				Word			
Halfword		Halfword		Halfword		Halfword	
Byte	Byte	Byte	Byte	Byte	Byte	Byte	Byte
0	1	2	3	4	5	6	7

https://student.cs.uwaterloo.ca/~isg/res/mips/opcodes

s2 s1

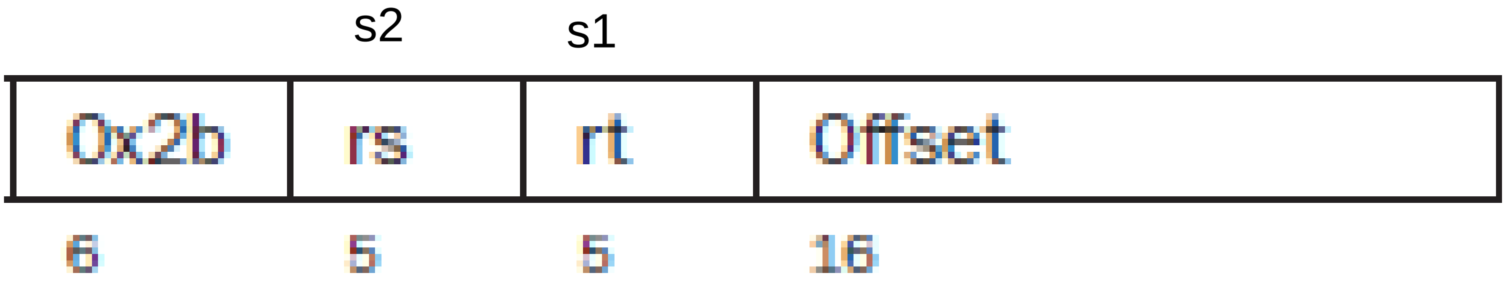


0x23	rs	rt	Offset
6	5	5	16

Memory Reference Instruction - sw

- Store Instruction
- copies data from register to memory
- Format : sw reg, address
- Example : sw $s1, 100($s2)

memory[100 +$s2]← $s1



	s2	s1	
0x2b	rs	rt	Offset
6	5	5	16

Arithmetic Instructions :add

add des, src1, src2

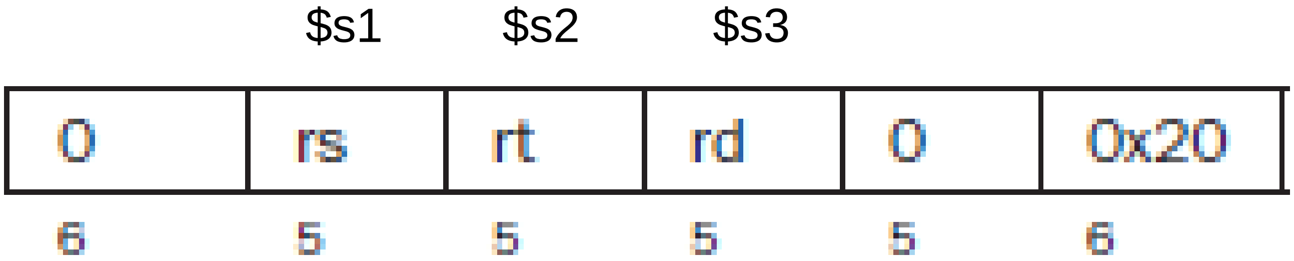
Addressing mode: register

Example: add $s3, $s1, $s2

meaning : $s3 = $s1 + $s2

opcode : 0 , function: 0x20

sa :0



	$s1	$s2	$s3		
0	rs	rt	rd	0	0x20
6	5	5	5	5	6

The opcode is the machine code representation of the instruction mnemonic. The opcode field is 6 bits long (bit 26 to bit 31) but always set to 0 in the R format. The actual instruction to perform is placed in the funct field.

Arithmetic Instructions :sub

sub des, src1, src2

Addressing mode: register

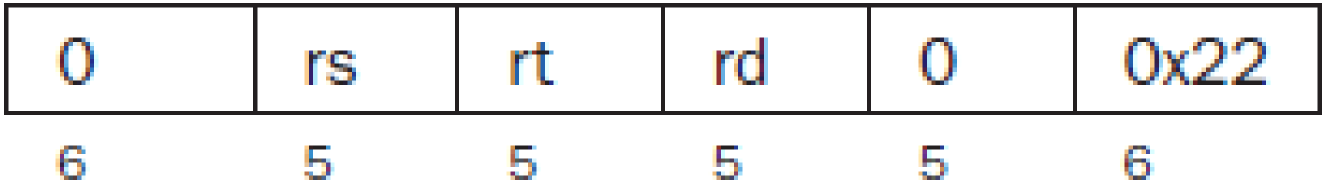
Example: sub $s3, $s1, $s2

meaning : $s3 = $s1 - $s2

opcode : 0 , function: 0x22

sa :0

$s1 $s2 $s3



0	rs	rt	rd	0	0x22
6	5	5	5	5	6

Logical instructions :and

and des, src1, src2

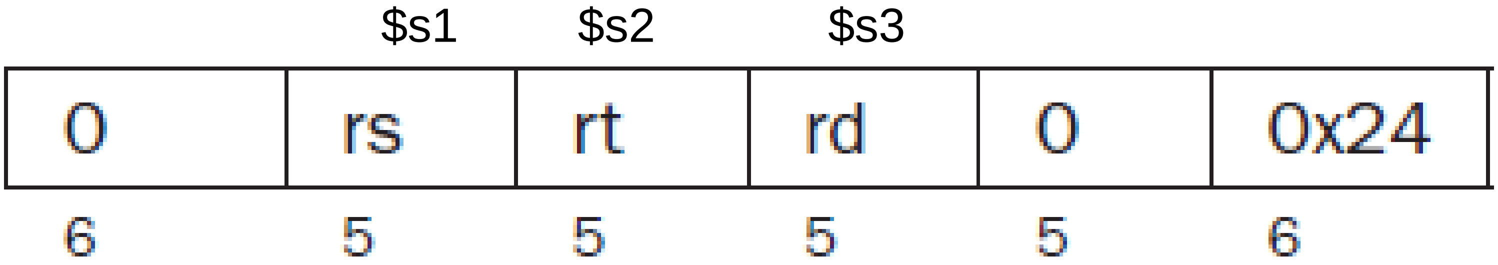
Addressing mode: register

Example: and $s3, $s1, $s2

meaning : $s3 = $s1 & $s2 (bit by bit)

opcode : 0 , function: 0x24

sa :0



	$s1	$s2	$s3		
0	rs	rt	rd	0	0x24
6	5	5	5	5	6

Logical instructions :or

or des, src1, src2

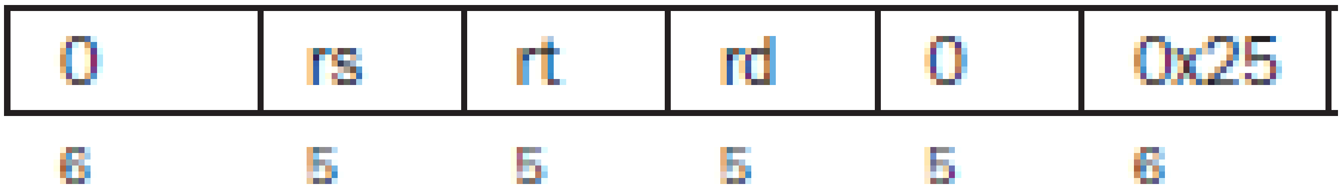
Addressing mode: register

Example: or $s3, $s1, $s2

meaning : $s3 = $s1 | $s2 (bit by bit)

opcode : 0 , function: 0x25

sa :0



	$s1	$s2	$s3		
0	rs	rt	rd	0	0x25
6	5	5	5	5	6

Logical instructions :slt

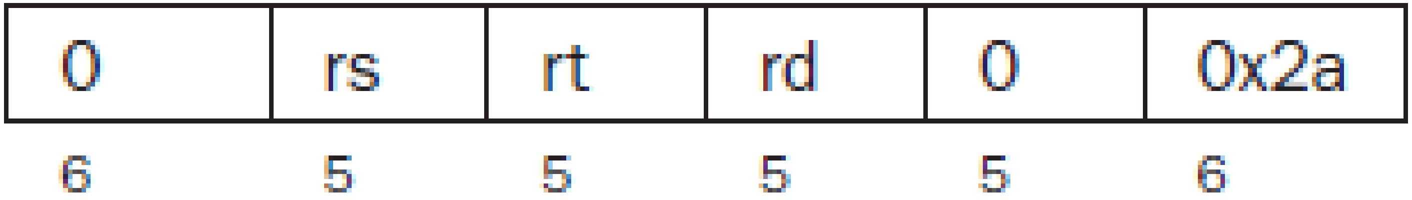
set on less than : slt

Format: slt des, src1, src2

set des = 1, if src1 < src2

Example:

slt $s3, $s1, $s2



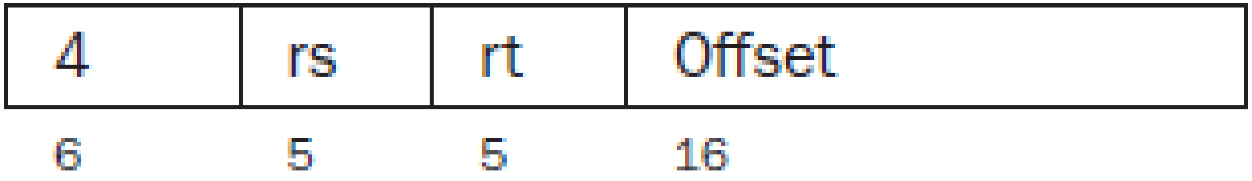
	$s1	$s2	$s3		
0	rs	rt	rd	0	0x2a
6	5	5	5	5	6

Set register rd to 1 if rs is less than rt, and to 0 otherwise

Branch Instructions: beq and j

beq : branch if equal

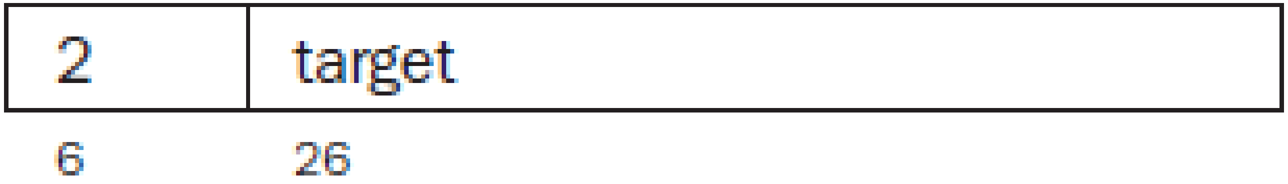
- format : beq $s1, $s2, label
- If $s1 = $s2 then branch to address specified as Label



4	rs	rt	Offset
6	5	5	16

j : jump unconditional

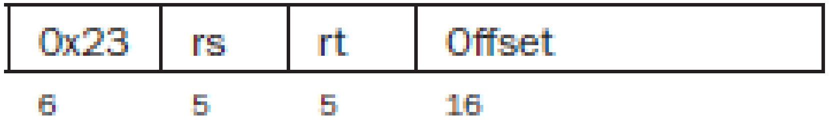
- format : j label



2	target
6	26

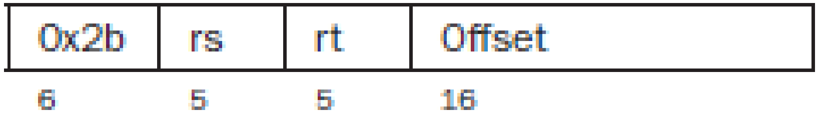
Summary

- lw $s1, 100($s2)



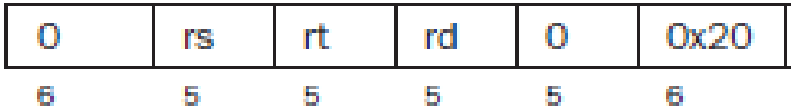
0x23	rs	rt	Offset
6	5	5	16

- sw $s1, 100($s2)



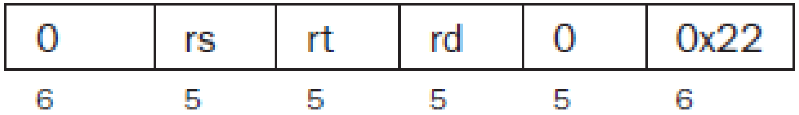
0x2b	rs	rt	Offset
6	5	5	16

- add $s1, $s2, $s3



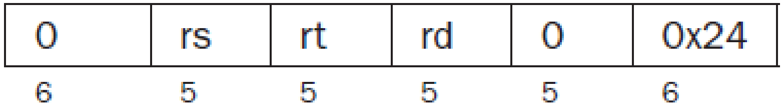
0	rs	rt	rd	0	0x20
6	5	5	5	5	6

- sub $s1, $s2, $s3



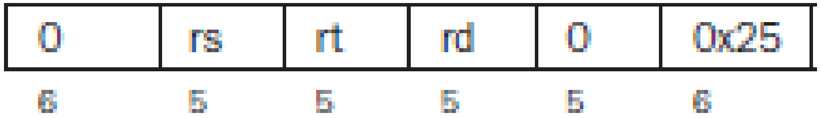
0	rs	rt	rd	0	0x22
6	5	5	5	5	6

- and $s1, $s2, $s3



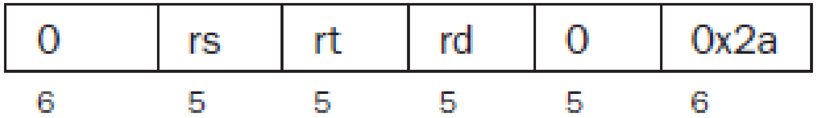
0	rs	rt	rd	0	0x24
6	5	5	5	5	6

- or $s1, $s2, $s3



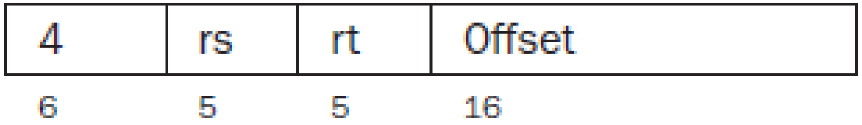
0	rs	rt	rd	0	0x25
6	5	5	5	5	6

- slt $t0, $s1, $s2



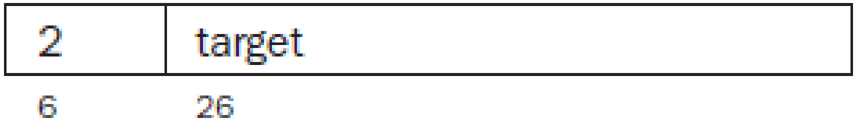
0	rs	rt	rd	0	0x2a
6	5	5	5	5	6

- beq $s1, $s2, label



4	rs	rt	Offset
6	5	5	16

- j label



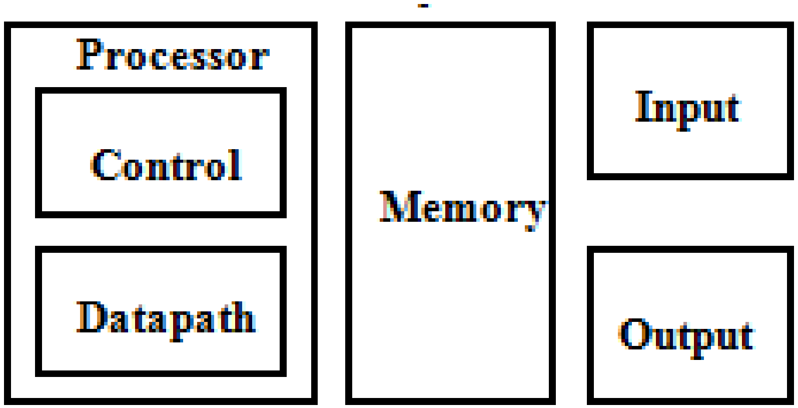
2	target
6	26

BITS Pilani
Pilani Campus

Data Path Design

Introduction

- The Five Classic Components of a Computer
  - Datapath: The component of the processor that performs data processing operations
  - Control: The component of the processor that commands the datapath, memory and I/O devices according to the instructions of the memory
  - Memory
  - Input device
  - Output device

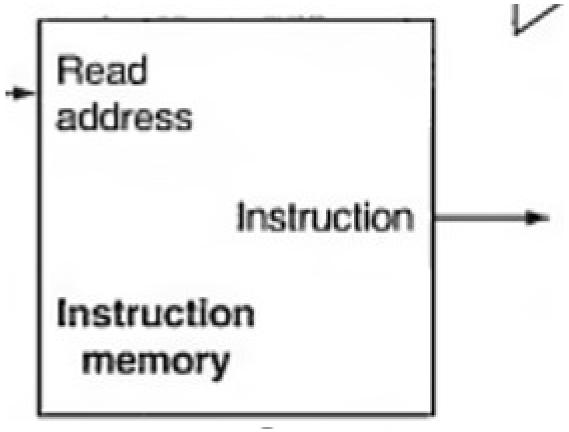

Processor
Control
Datapath
Memory
Input
Output

Building a datapath

- Datapath elements :memory – instruction/data, register file, ALU, Adders, Multiplexers....
- Demonstrate Datapath implementation for the following instruction:
    - lw and sw
    - add, sub, and, or and slt
    - beq and j
- Instruction Cycle = Fetch + Execute

Lets Do this…

write a datapath used for fetching instructions and incrementing the program counter


Read address
Instruction
Instruction memory

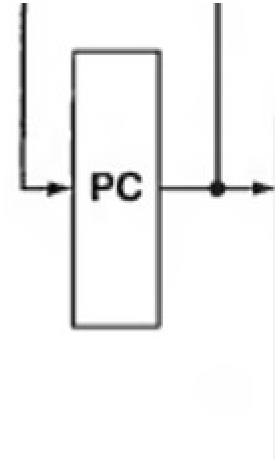
Step number 1
Instruction memory

It has instructions

Given instruction address it gives instruction

Lets Do this…

write a datapath used for fetching instructions and incrementing the program counter



Step number 2
PC – Program counter is a 32 bit register

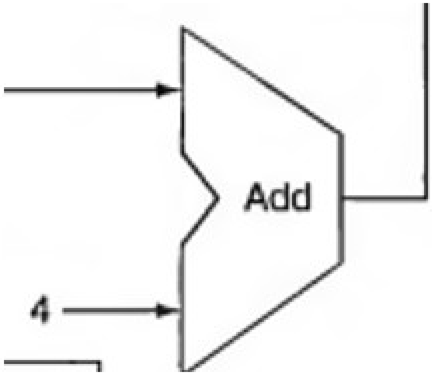
It has memory address of where the program is

Updated after every clock cycle

Readable and writeable

Lets Do this…

write a datapath used for fetching instructions and incrementing the program counter



Step number 3
Adder

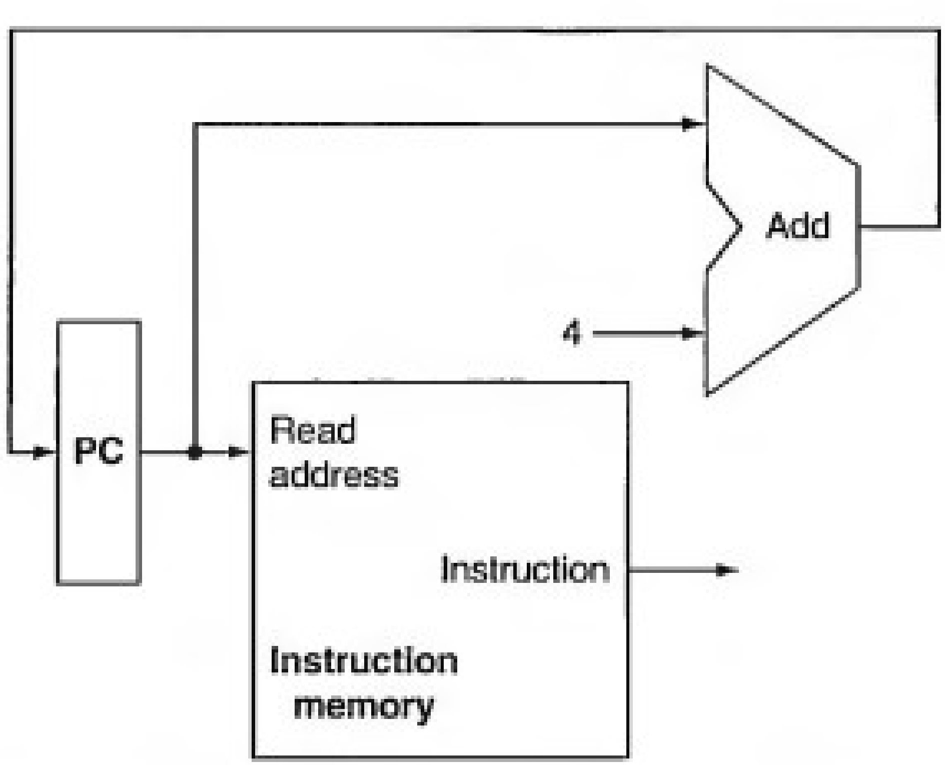
It takes two input values and gives an output

Takes PC and 4 (as address is 32 bits, 4 bytes) and gives output

Output is the next instruction address

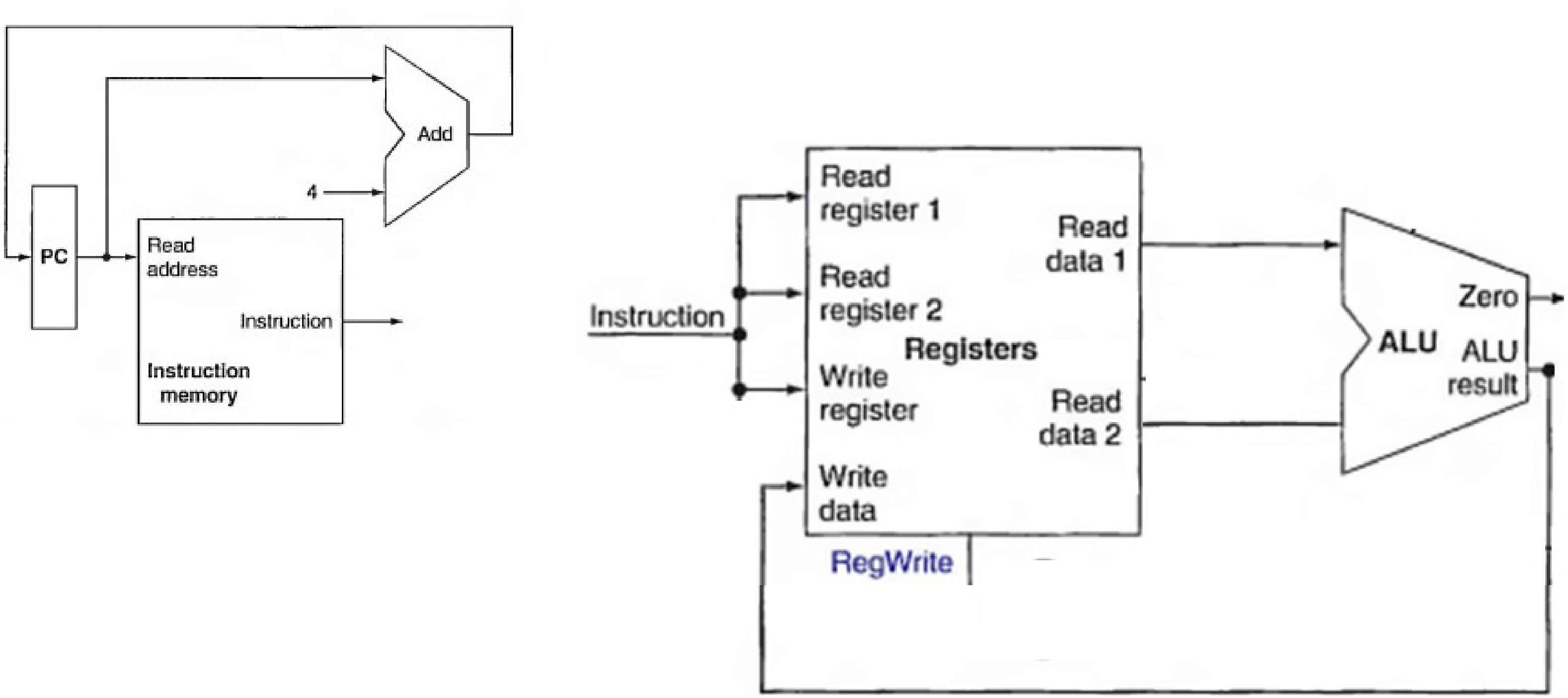
Lets Do this…

write a datapath used for fetching instructions and incrementing the program counter


Add
4
PC
Read address
Instruction
Instruction memory

Steps – Data path for R-Type

1. Get instruction from Program Counter


Add
4
PC
Read address
Instruction
Instruction memory
Instruction
Read register 1
Read register 2
Write register
Write data
Registers
Read data 1
Read data 2
RegWrite
ALU
Zero
ALU result

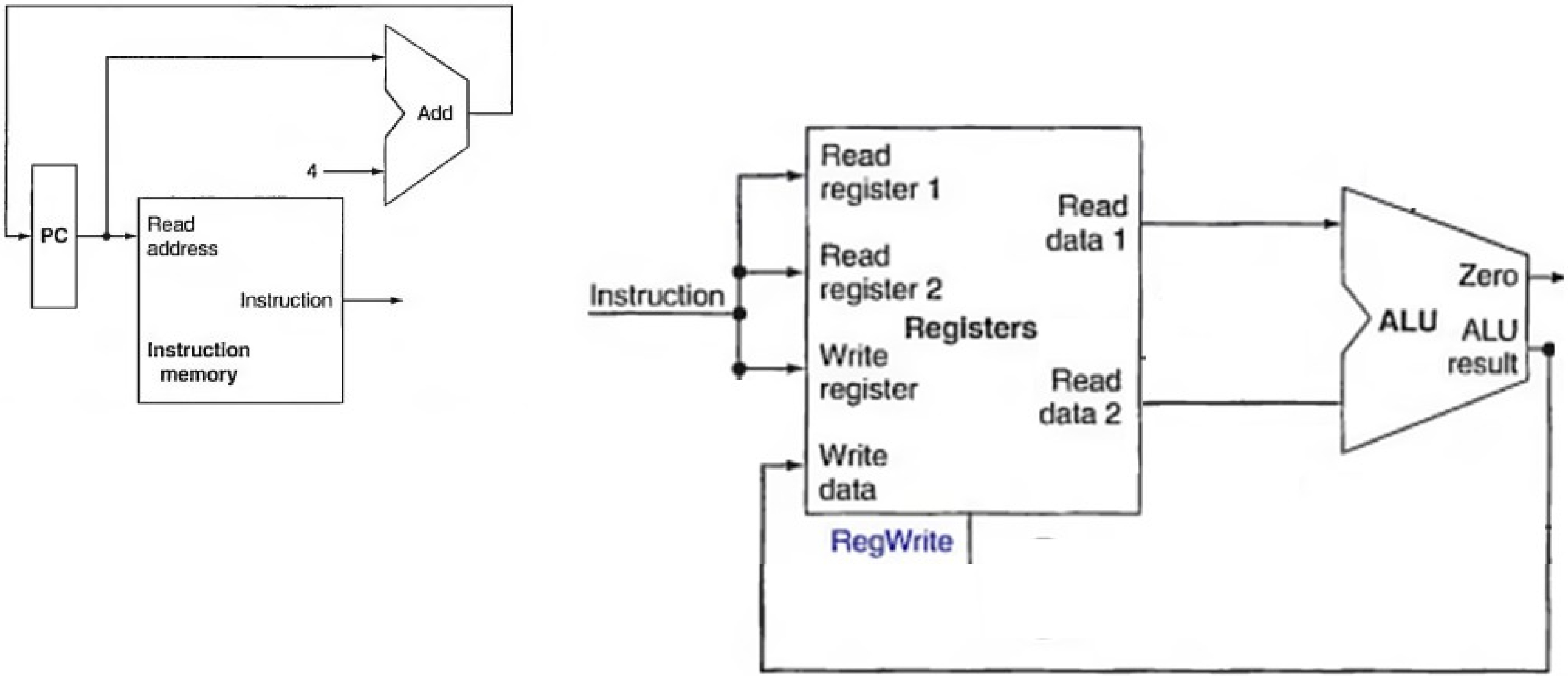
The datapath for the memory instructions and the R-type instructions.

31 26 21 16 11 6 0

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Steps – Data path for R-Type

2. Fetch instruction from Instruction memory
3. Decode Instruction

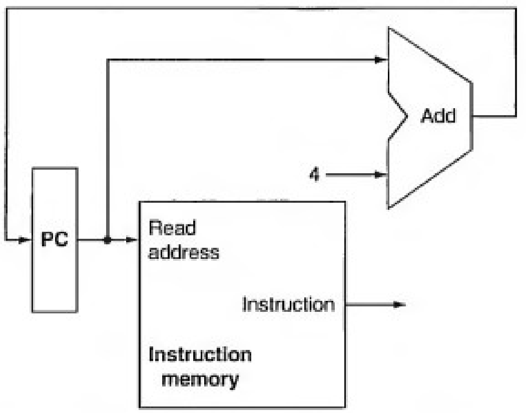

Add
4
PC
Read address
Instruction
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
RegWrite
ALU
Zero
ALU result

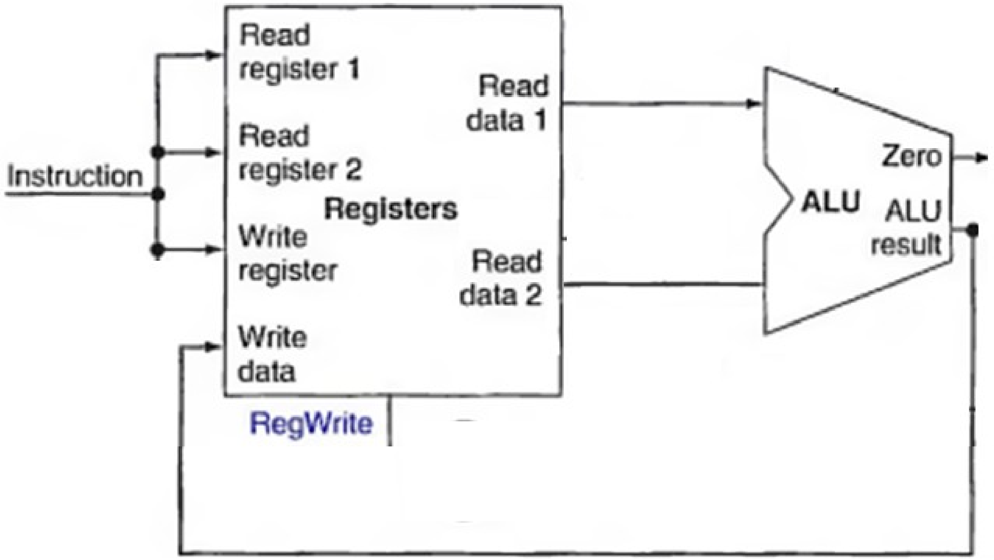
31 26 21 16 11 6 0 ıs and the R-type instructions.

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Steps – Data path for R-Type

4. Pass rs, rt and rd into read register and write register arguments

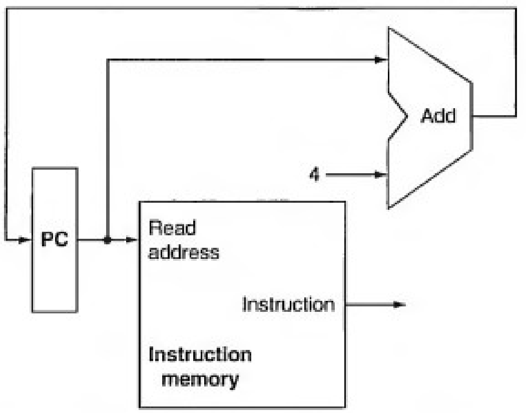

Add
4
PC
Read
address
Instruction
Instruction
memory

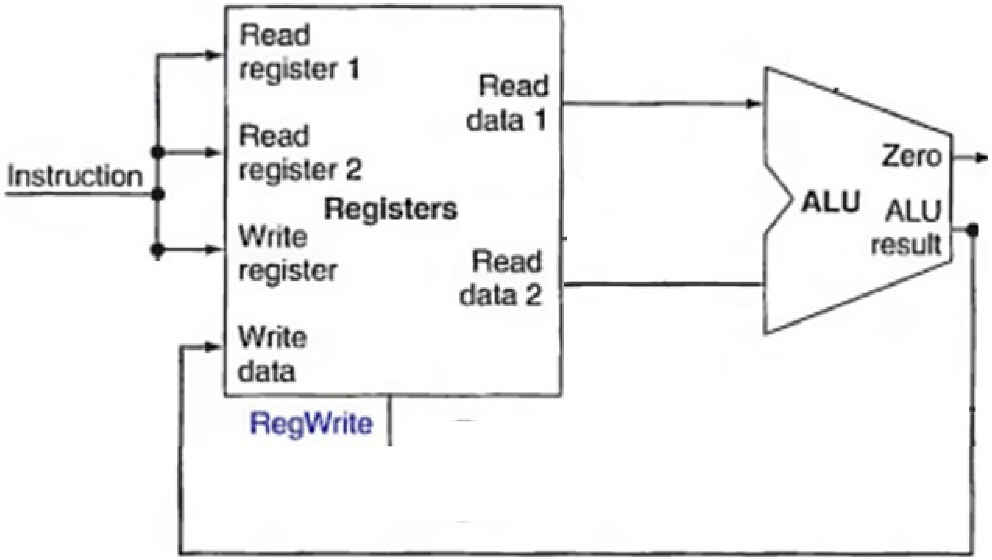

Instruction
Read
register 1
Read
register 2
Registers
Write
register
Write
data
Read
data 1
Read
data 2
RegWrite
ALU
Zero
ALU
result

The datapath for the memory instructions and the R-type instructions.

Steps – Data path for R-Type

5. Retrieve Data from read register 1, read register 2(rs,rt)

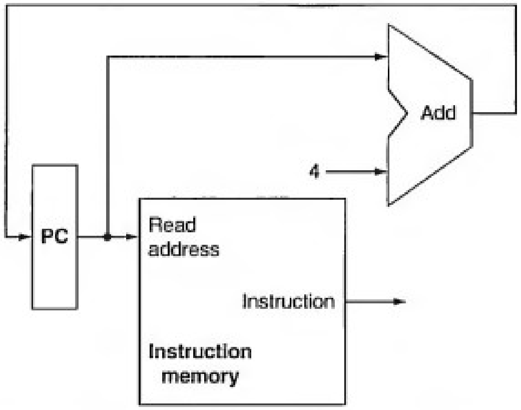

Add
4
PC
Read address
Instruction
Instruction memory

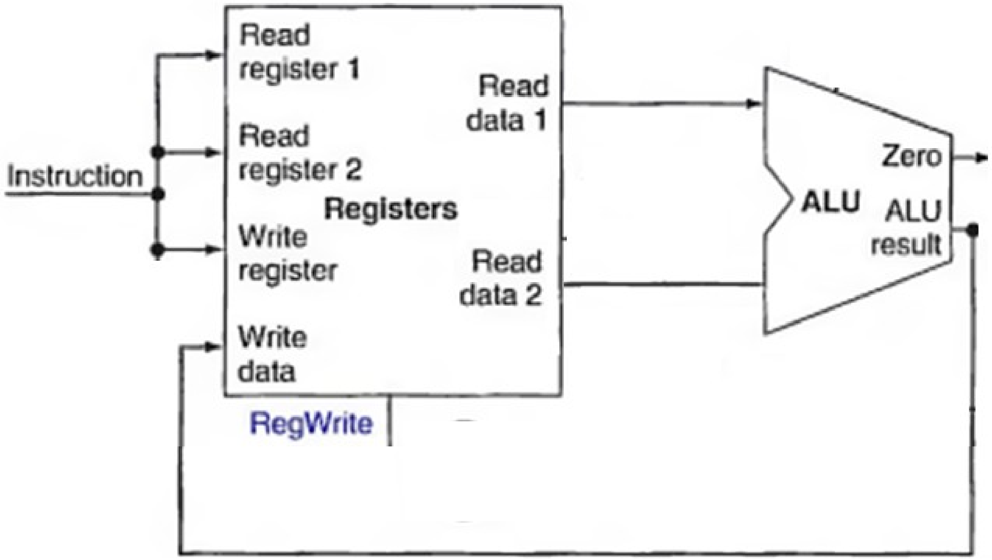

Instruction
Read register 1
Read register 2
Write register
Write data
Registers
Read data 1
Read data 2
RegWrite
ALU
Zero
ALU result

The datapath for the memory instructions and the R-type instructions.

Steps – Data path for R-Type

6. Pass contents of rs and rt into the ALU as operands to be performed.

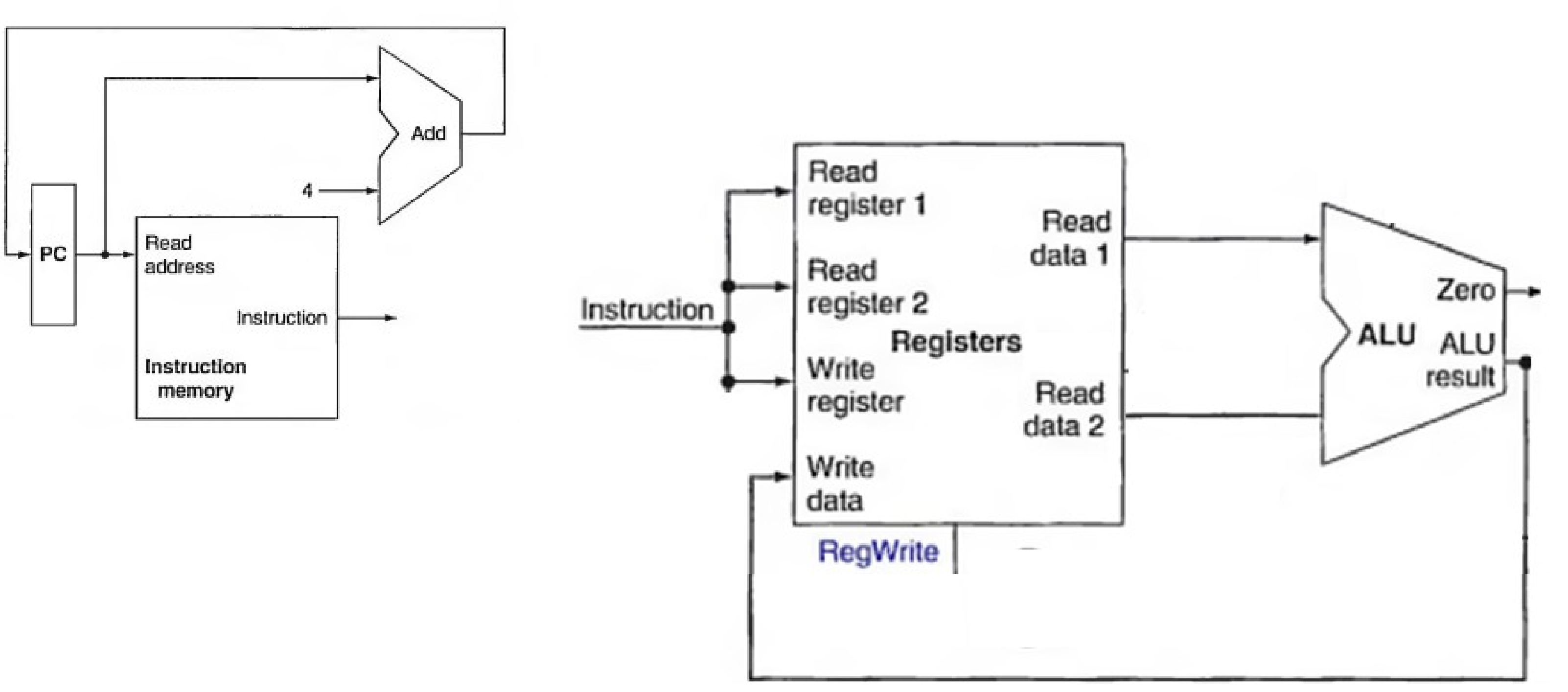

Add
4
PC
Read address
Instruction
Instruction memory


Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
RegWrite
ALU
Zero
ALU result

The datapath for the memory instructions and the R-type instructions.

Steps – Data path for R-Type

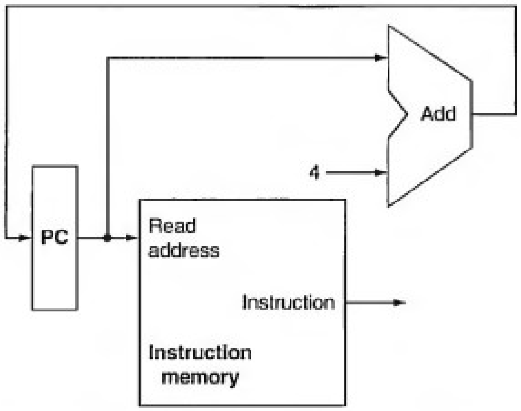
7. Retrieve result of operation performed by ALU and pass back as the write data argument of the register field (RegWrite enabled)

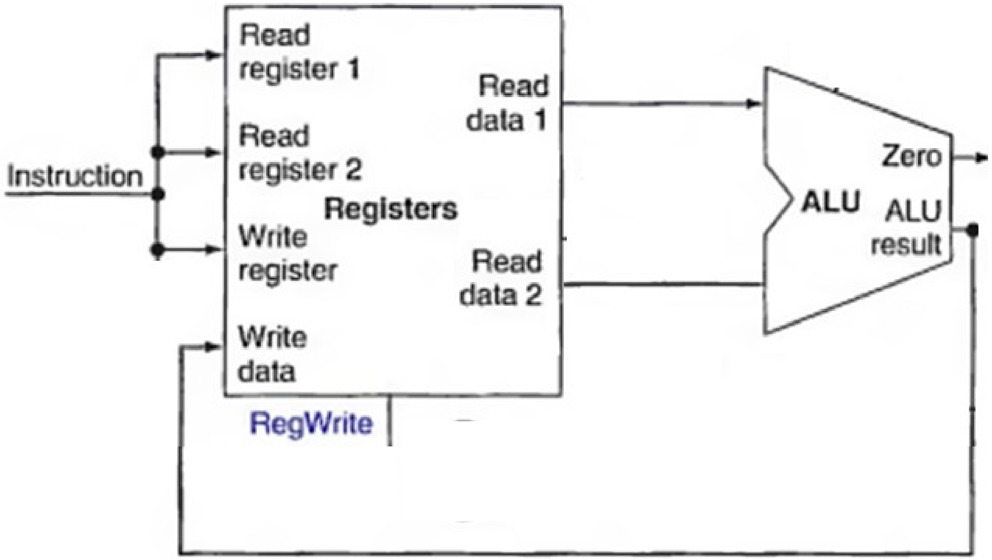

Add
4
PC
Read address
Instruction
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
RegWrite
ALU
Zero
ALU result

The datapath for the memory instructions and the R-type instructions.

Steps – Data path for R-Type

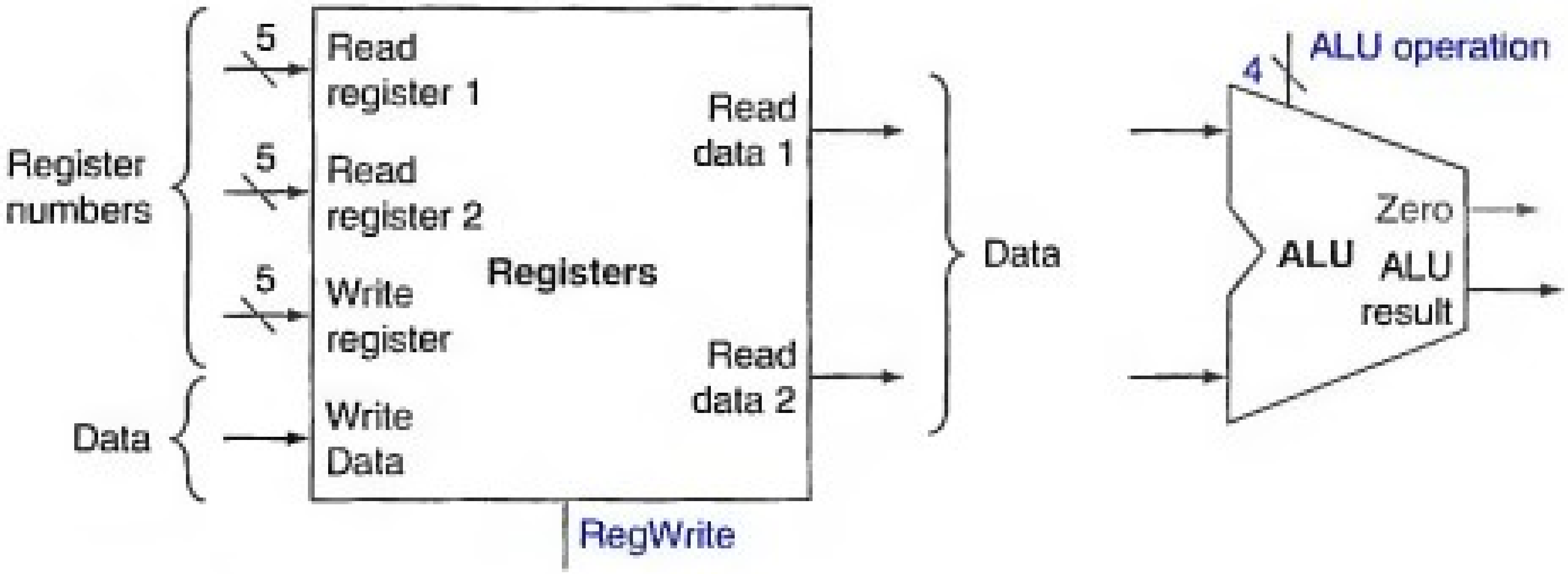
8. Add 4 bytes to the PC value to get the next instruction address


Add
4
PC
Read address
Instruction
Instruction memory


Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
RegWrite
ALU
Zero
ALU result

The datapath for the memory instructions and the R-type instructions.

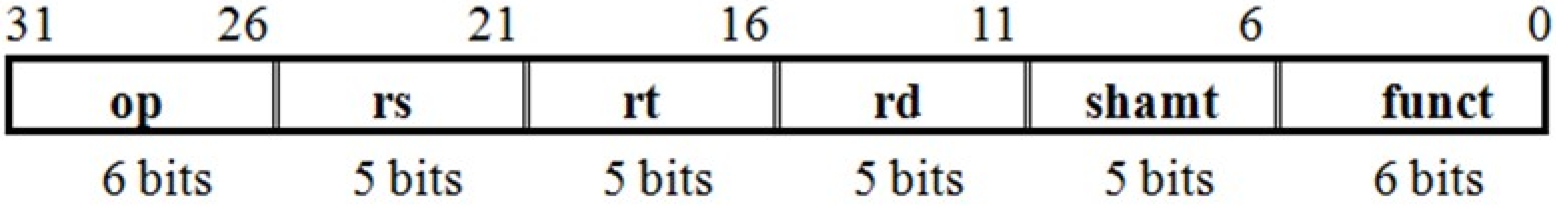
Data Path for R-type instruction


Register numbers
5
5
5
Read register 1
Read register 2
Write register
Write Data
Registers
Read data 1
Read data 2
Data
Data
RegWrite
ALU operation
4
ALU
Zero
ALU result

a. Registers

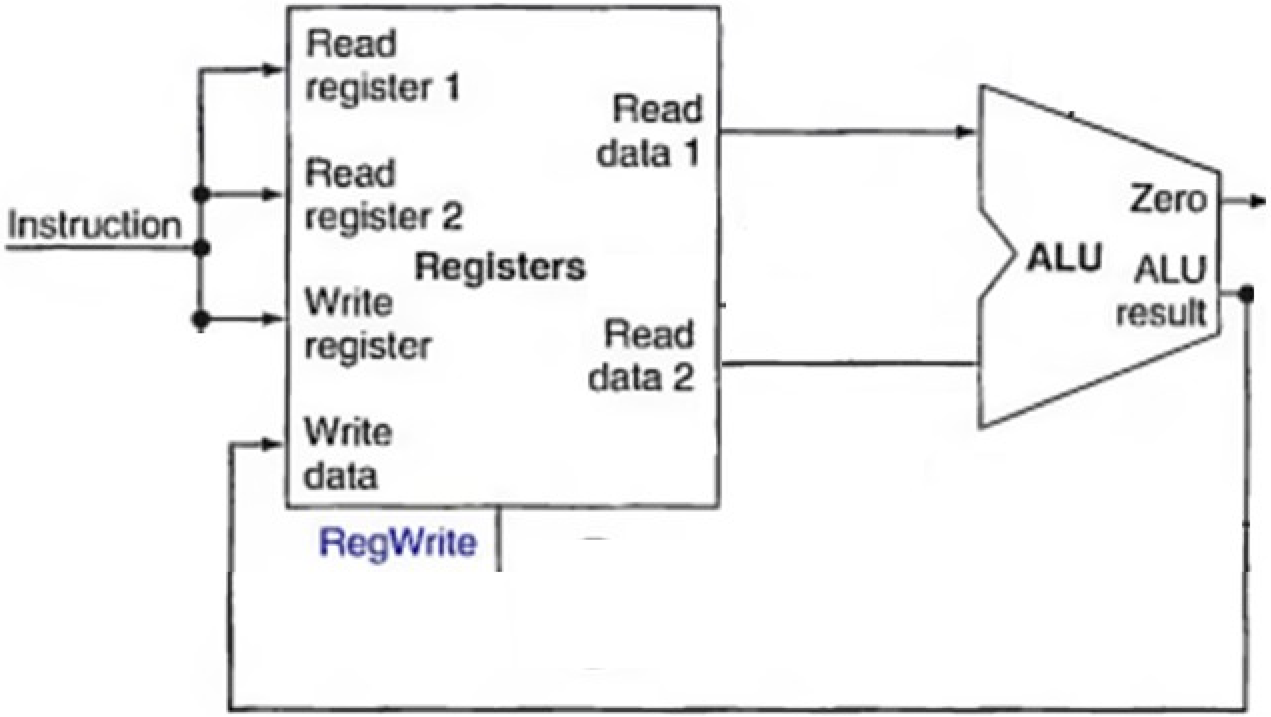
b. ALU

Examples:
add, sub, and, or, slt ➔ rs, rt, rd

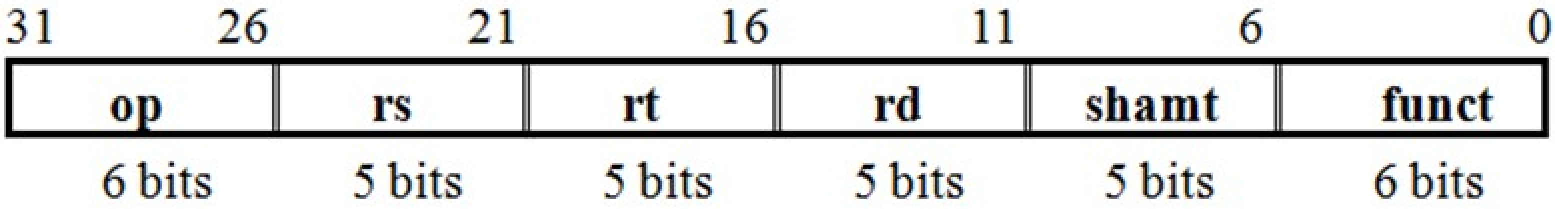


31 26	21	16	11	6	0
op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Data Path for R-type instruction…


Instruction
Read register 1
Read register 2
Write register
Write data
Registers
Read data 1
Read data 2
RegWrite
ALU
Zero
ALU result

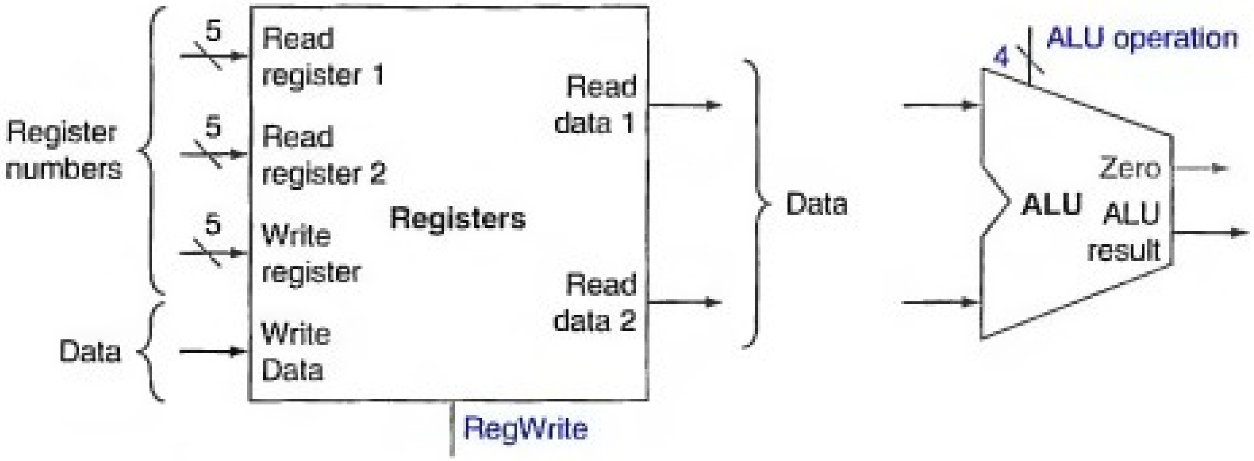
The datapath for the memory instructions and the R-type instructions.



31 26	21	16	11	6	0
op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

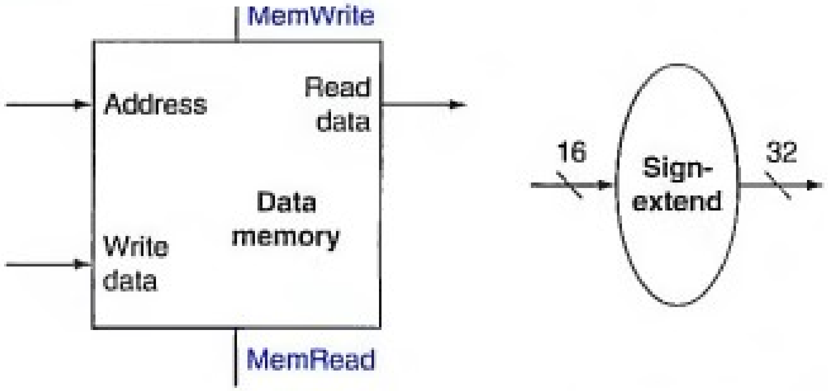
Data Path for lw and sw instructions

Instruction Summary


Register numbers
5
5
5
Read register 1
Read register 2
Write register
Data
Write Data
Registers
Read data 1
Read data 2
Data
RegWrite
ALU operation
4
ALU
Zero
ALU result

a. Registers

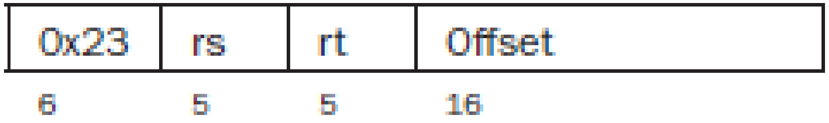
b. ALU


MemWrite
Address
Read data
Write data
Data memory
MemRead
16
Sign-extend
32

a. Data memory unit

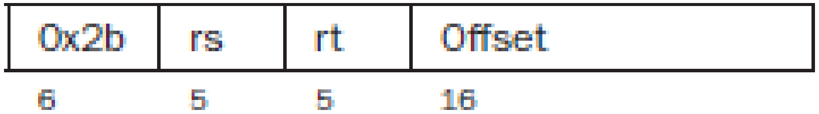
b. Sign extension unit

lw $s1, 100($s2)



0x23	rs	rt	Offset
6	5	5	16

sw $s1, 100($s2)



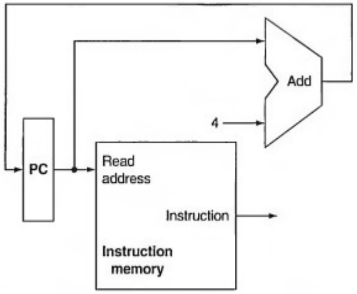
0x2b	rs	rt	Offset
6	5	5	16

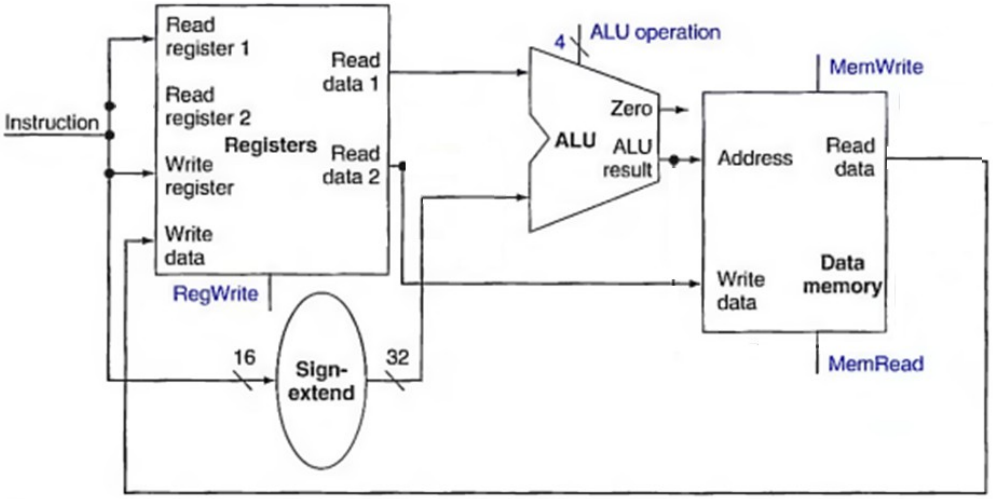
Data Path for lw instructions

Lw – Load word from memory
$s2 – Source register
$s1 – Destination register
100 – offset, sign extended
Sign extension done to fill higher order bits
$s1 <- Memory[100 +$s2]

lw $s1, 100($s2)

	s2	s1	
0x23	rs	rt	Offset
6	5	5	16


Add
4
PC
Read address
Instruction
Instruction memory


Instruction
Read register 1
Read register 2
Write register
Write data
Registers
Read data 1
Read data 2
RegWrite
16
Sign-extend
32
4
ALU operation
ALU
Zero
ALU result
MemWrite
Address
Read data
Write data
Data memory
MemRead

Two data path required
- Access memory
- Sign extension

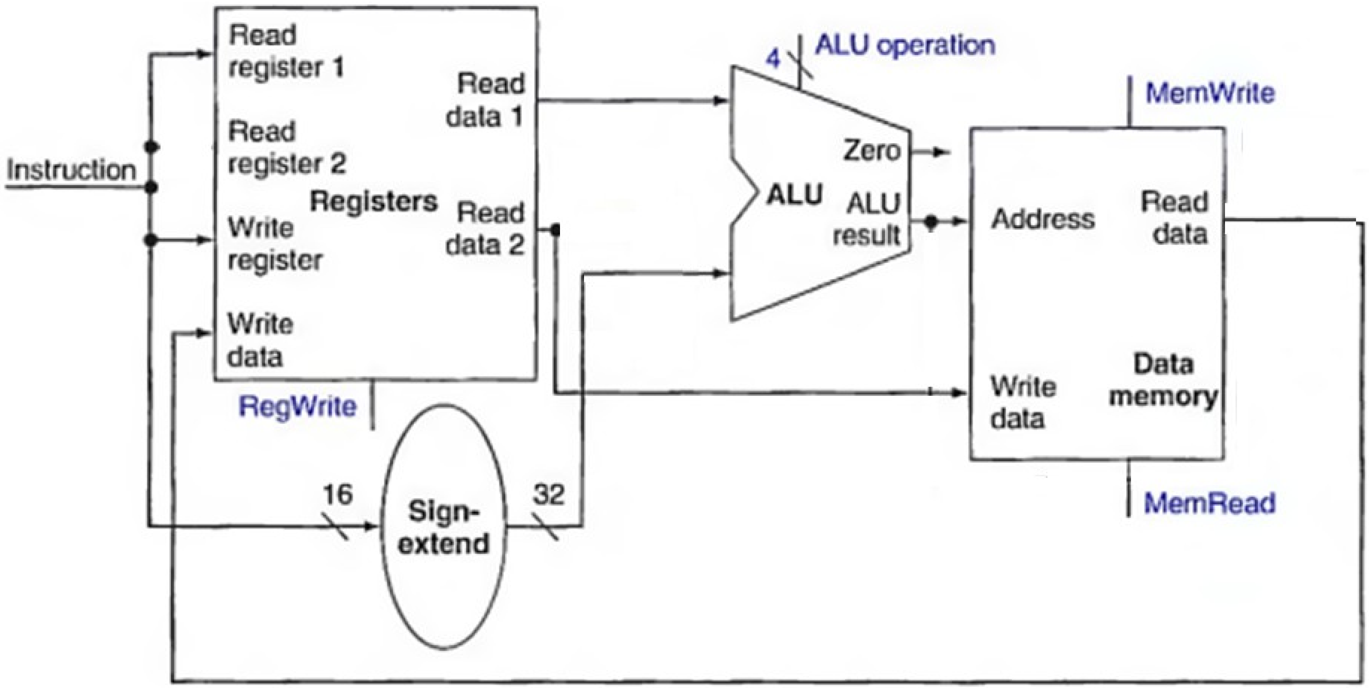
Data Path for lw instructions

There are two inputs

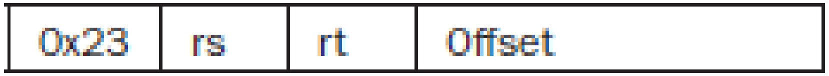
1. Address of memory location to access
2. Data to be written to memory

Control

1. MemWrite 2. MemRead


Instruction
Read register 1
Read register 2
Write register
Write data
Registers
Read data 1
Read data 2
RegWrite
16
Sign-extend
32
4
ALU operation
ALU
Zero
ALU result
MemWrite
Address
Write data
Read data
Data memory
MemRead

lw $s1, 100($s2)

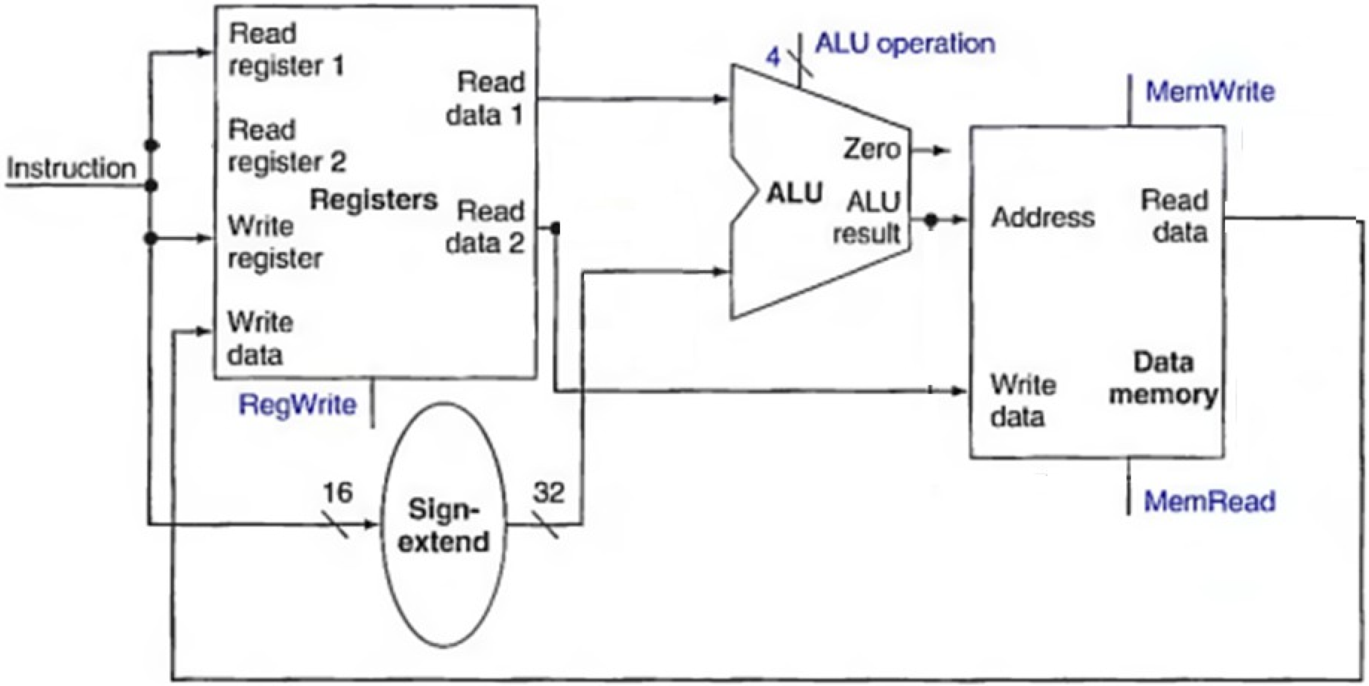


0x23	rs	rt	Offset

Data Path for lw instructions

Sign extension

We take most significant bit field of original field until we have reached the desired field width



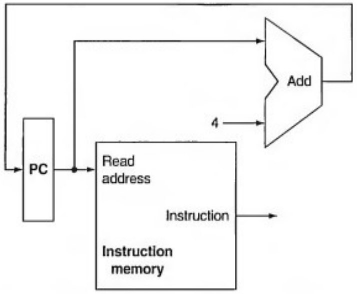
Read register 1
Read register 2
Write register
Write data
Registers
Read data 1
Read data 2
Instruction
RegWrite
16
Sign-extend
32
4
ALU operation
ALU
Zero
ALU result
MemWrite
Address
Read data
Write data
Data memory
MemRead

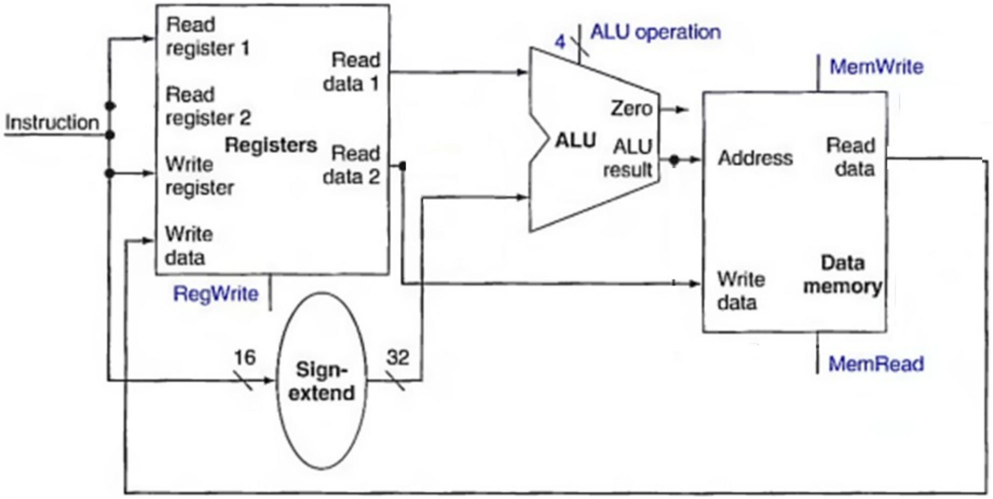
Data Path for lw instructions

$s1 <- Memory[100 +$s2]

lw $s1, 100($s2)

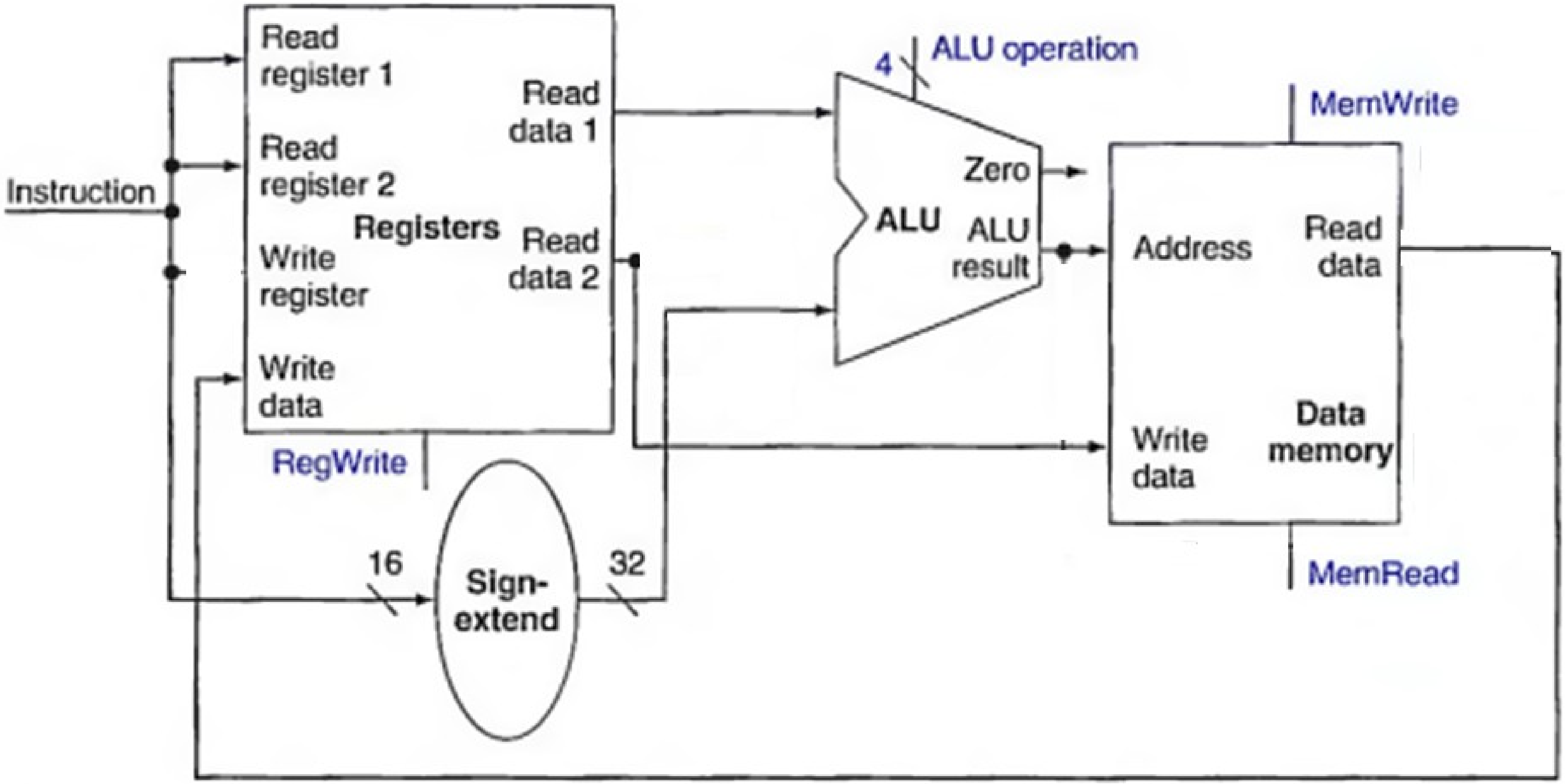
	s2	s1	
0x23	rs	rt	Offset
6	5	5	16


Add
4
PC
Read address
Instruction
Instruction memory


Instruction
Read register 1
Read register 2
Write register
Write data
Registers
Read data 1
Read data 2
RegWrite
16
Sign-extend
32
4
ALU operation
ALU
Zero
ALU result
MemWrite
Address
Read data
Write data
Data memory
MemRead

1.Extend sign
2. Obtain address after computing using ALU
3. Provide address
4. Read Data from memory
5. Load that into register
6. PC operations like before

Data Path for sw instructions


Read register 1
Read register 2
Write register
Write data
Registers
Read data 1
Read data 2
Instruction
RegWrite
16
Sign-extend
32
4
ALU operation
ALU
Zero
ALU result
MemWrite
Address
Read data
Write data
Data memory
MemRead

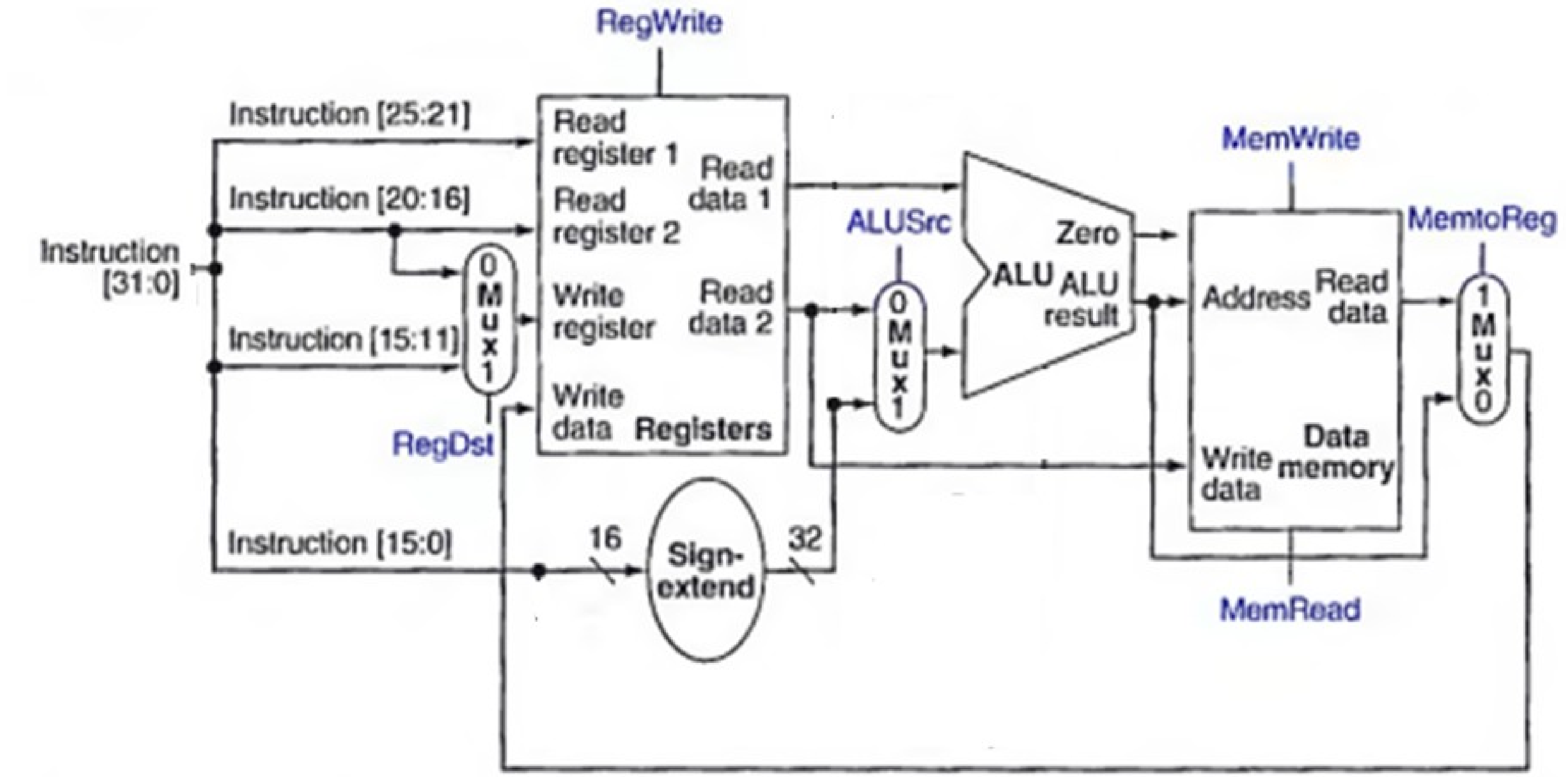
sw $s1, 100($s2)

Memory[100 +$s2] <- $s1

1. Extend sign
2. Obtain address after adding in ALU
3. Provide address
4. Store data from register to memory in address from step 3
5. Program counter operations as before

0x2b	rs	rt	Offset
6	5	5	16

Combined Datapath for R-Type and I-Type instructions


RegWrite
Instruction [25:21]
Instruction [20:16]
Instruction [31:0]
Instruction [15:11]
Instruction [15:0]
0 Mux 1
RegDst
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
16
Sign-extend
32
ALUSrc
0 Mux 1
Zero
ALU
ALU result
MemWrite
Address
Read data
Write data
Data memory
MemRead
MemtoReg
1 Mux 0

The datapath with all necessary multiplexors and all control lines identified.

R-Type

0	rs	rt	rd	0	0x20
6	5	5	5	5	6

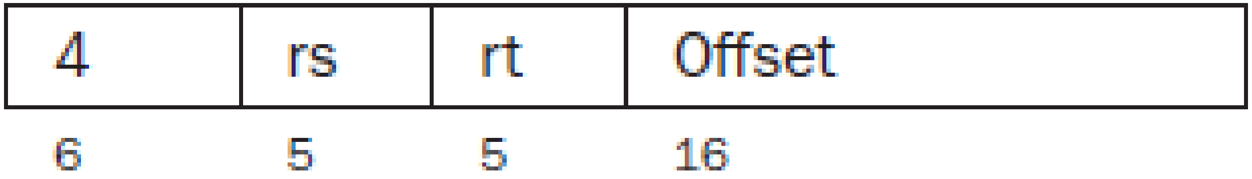
I-Type

0x2b	rs	rt	Offset
6	5	5	16

BEQ instruction

beq : branch if equal

- format : beq \$s1, \$s2, label
- If \$s1 = \$s2 then branch to address specified as Label/offset

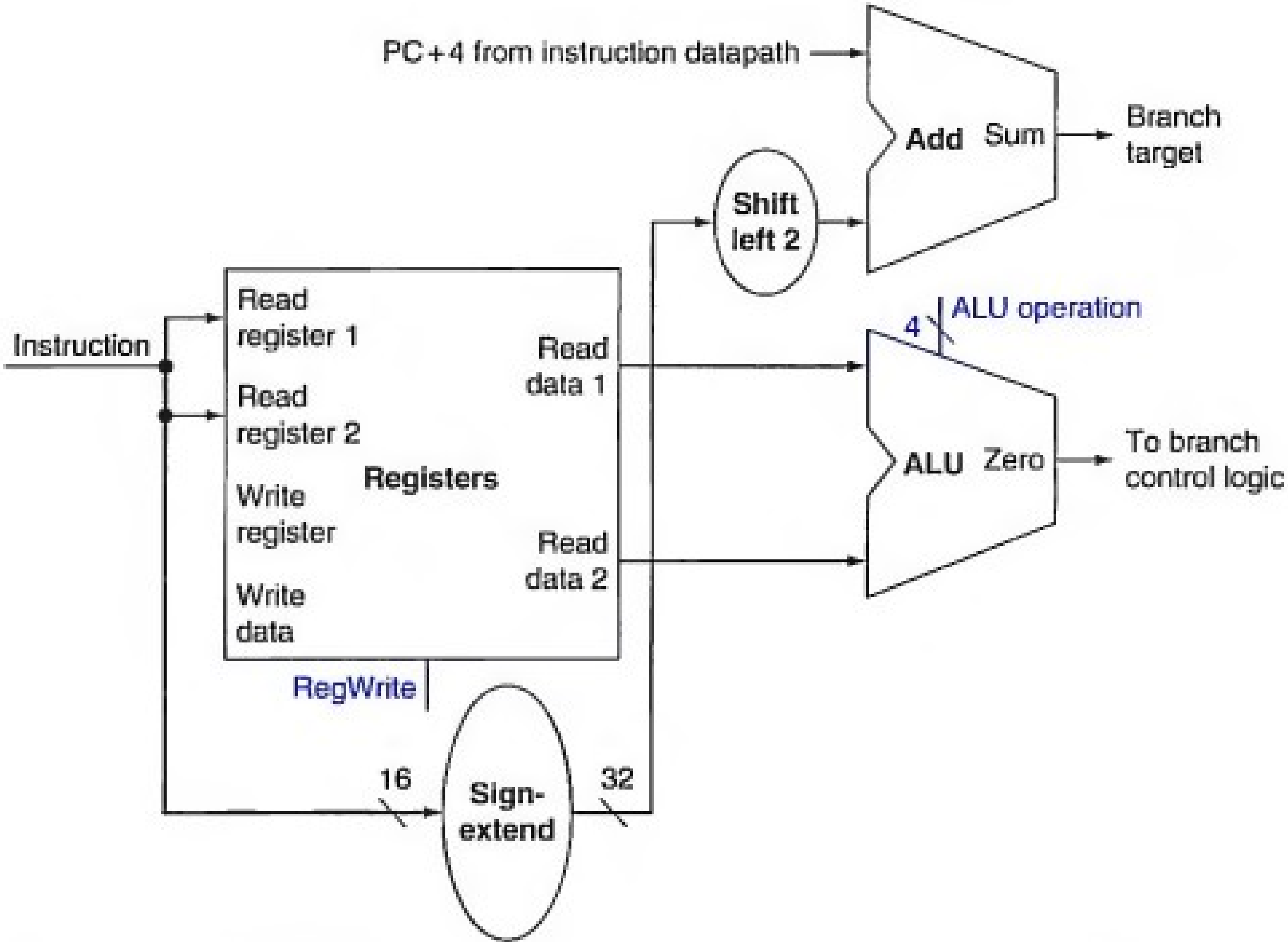


4	rs	rt	Offset
6	5	5	16

beq instruction

- beq $s1, $s2, offset
- Two important points to be noted
  - The instruction set architecture defines that the **base for the branch address** calculation is the address of the instruction following the branch
  - the architecture also states that the offset field is **shifted left 2 bits;** offset specified in immediate is **words** and not **bytes;** shifting left by 2bits gets that result
- branch taken or branch not taken

Datapath for beq instruction


PC+4 from instruction datapath
Add
Sum
Branch target
Shift left 2
4
ALU operation
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
RegWrite
ALU
Zero
To branch control logic
16
Sign-extend
32

Jump instruction

j <26 bit jump offset>

the jump instruction operates by replacing the lower 28 bits of the PC with the lower 26 bits of the instruction shifted left by 2 bits

Creating a single cycle datapath

Simplest datapath executes all instructions in one clock cycle

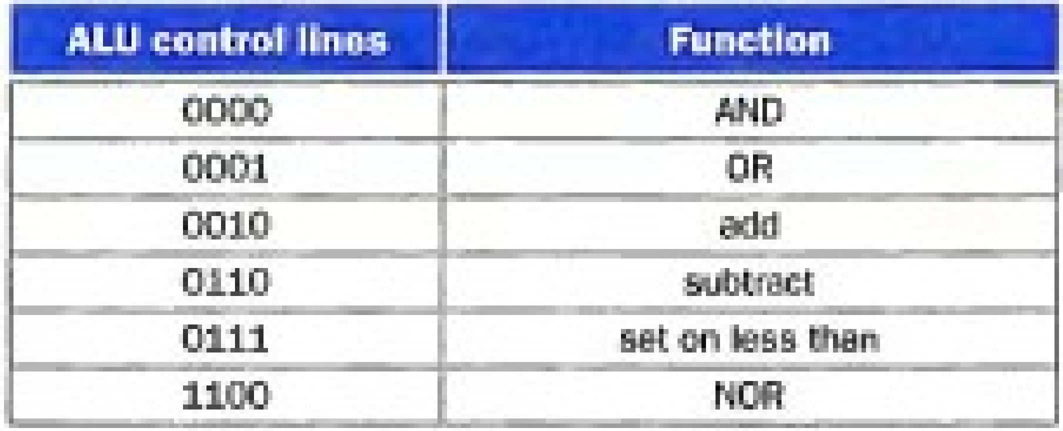
- No element in the datapath can be used more than once
- Any element needed more than once must be duplicated

Control unit design

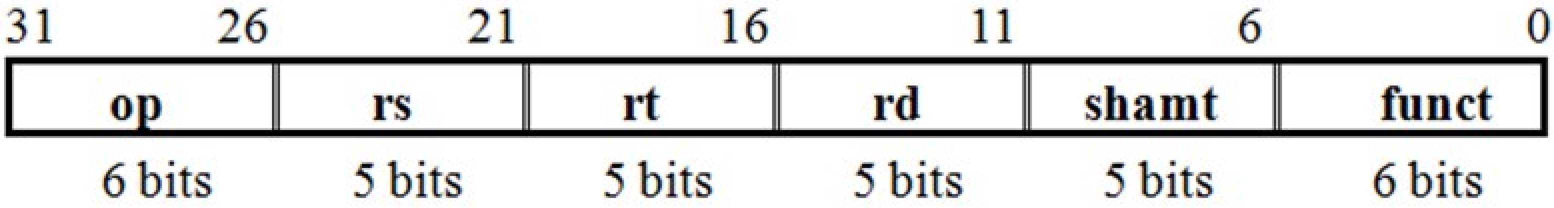
Previous

The ALU control

- used by load and store instructions
- used by r-type instruction : AND, OR, add, sub and slt
  - opcode : zero
  - operation performed based on “function field”



ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

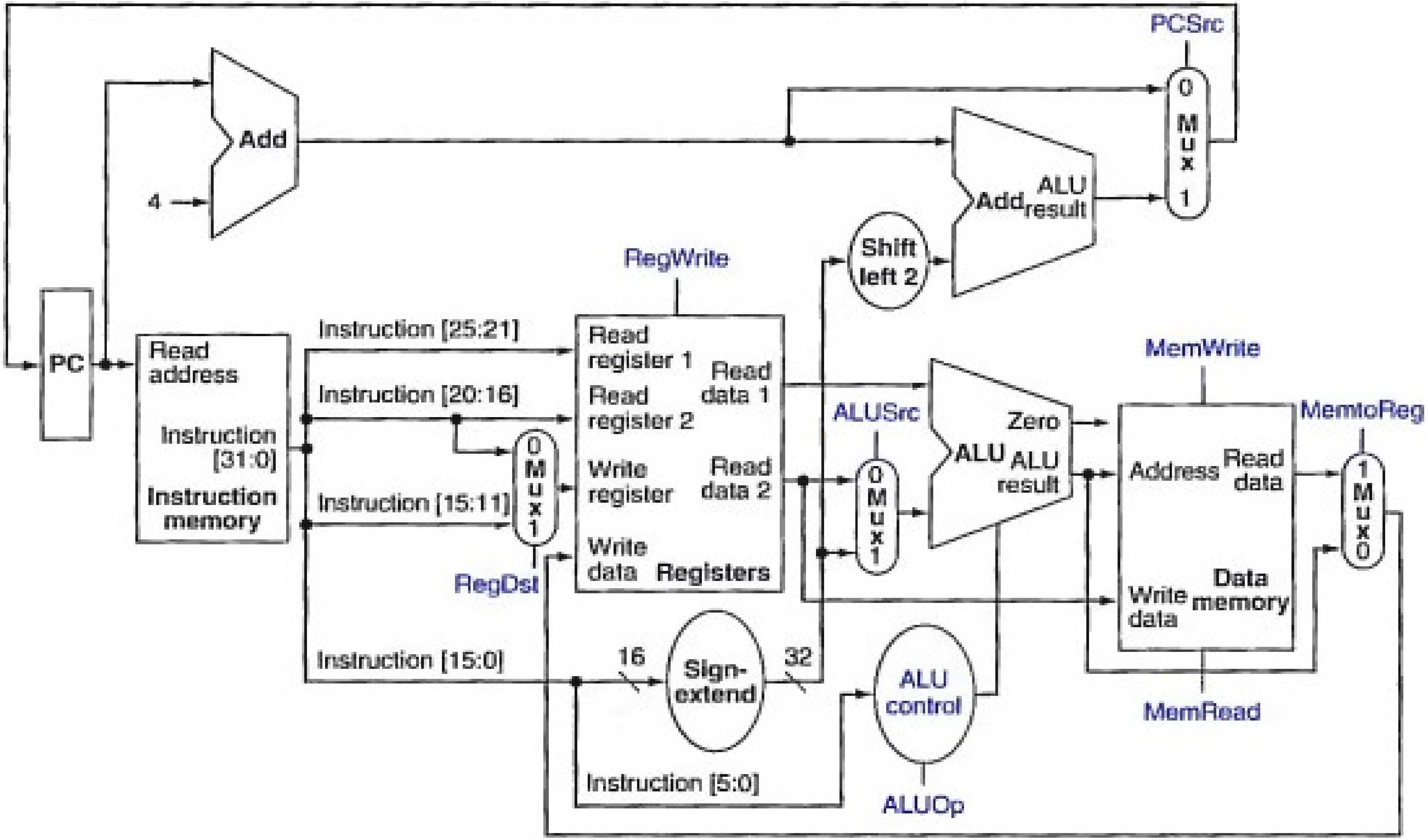


31	26	21	16	11	6	0
op	rs	rt	rd	shamt	funct	
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	

Contd…

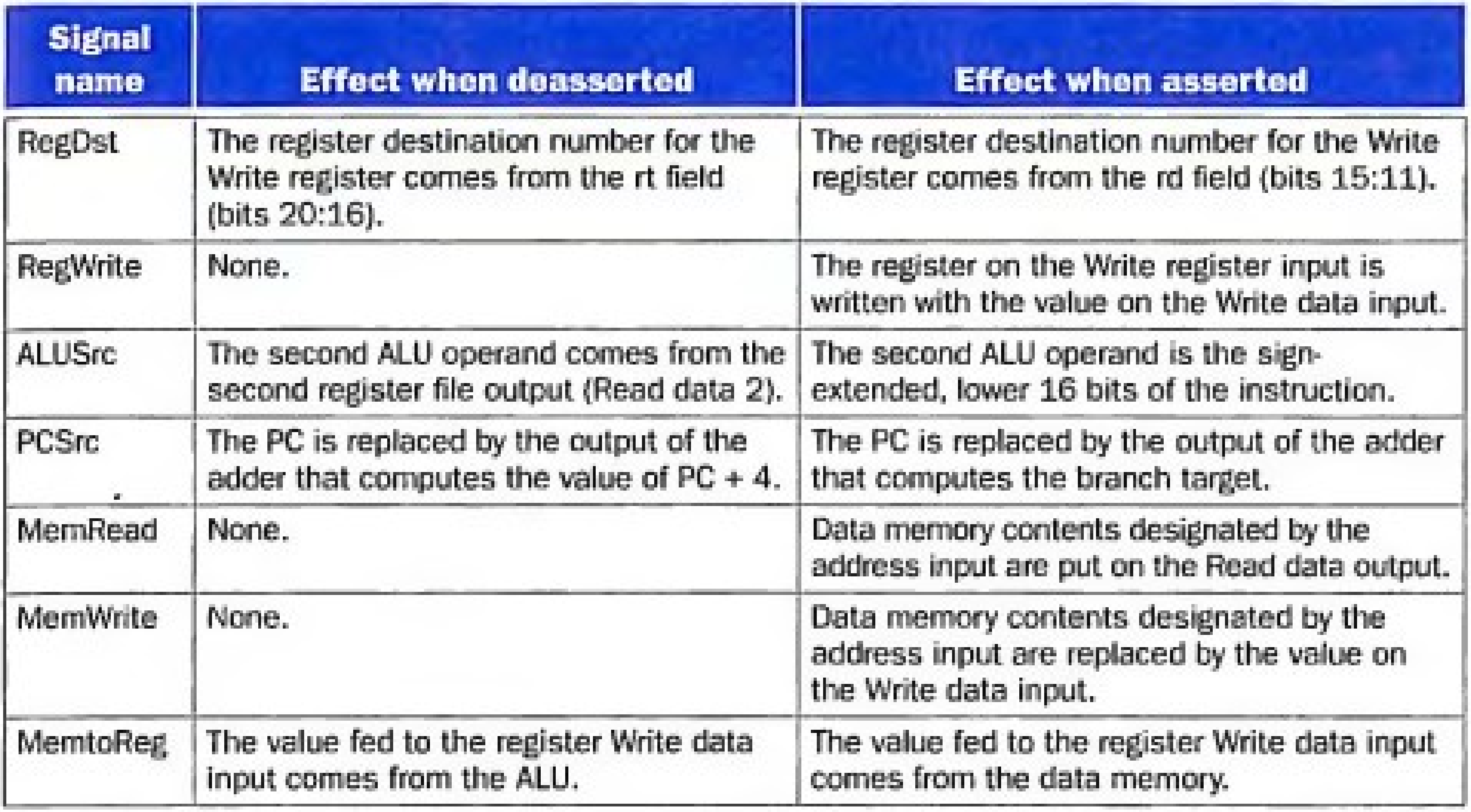
- ALU control unit generates 4 bit ALUOperation control signal based on
  - function field
  - 2 bit control field → ALUOp
    - 00 → lw and sw
    - 01→ beq (sub)
    - 10→ add, subtract, AND, OR and slt
- Main Control unit generates 7 control signals : RegDst, RegWrite, ALUSrc, PCSrc, MemRead, MemWrite and MemtoReg

Datapath


PCSrc
Add
4
M
u
x
0
1
ALU
Add result
Shift left 2
RegWrite
PC
Read address
Instruction [31:0]
Instruction memory
Instruction [25:21]
Instruction [20:16]
Instruction [15:11]
Instruction [15:0]
RegDst
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
ALUSrc
Zero
ALU
ALU result
MemWrite
MemtoReg
Address
Read data
Write data
Data memory
MemRead
16
32
Sign-extend
ALU control
ALUOp
Instruction [5:0]

The datapath with all necessary multiplexors and all control lines identified.

Effects of 7 control signals



Signal name	Effect when deasserted	Effect when asserted
RegDst	The register destination number for the Write register comes from the rt field (bits 20:16).	The register destination number for the Write register comes from the rd field (bits 15:11).
RegWrite	None.	The register on the Write register input is written with the value on the Write data input.
ALUSrc	The second ALU operand comes from the second register file output (Read data 2).	The second ALU operand is the sign-extended, lower 16 bits of the instruction.
PCSrc	The PC is replaced by the output of the adder that computes the value of PC + 4.	The PC is replaced by the output of the adder that computes the branch target.
MemRead	None.	Data memory contents designated by the address input are put on the Read data output.
MemWrite	None.	Data memory contents designated by the address input are replaced by the value on the Write data input.
MemtoReg	The value fed to the register Write data input comes from the ALU.	The value fed to the register Write data input comes from the data memory.

Major observations

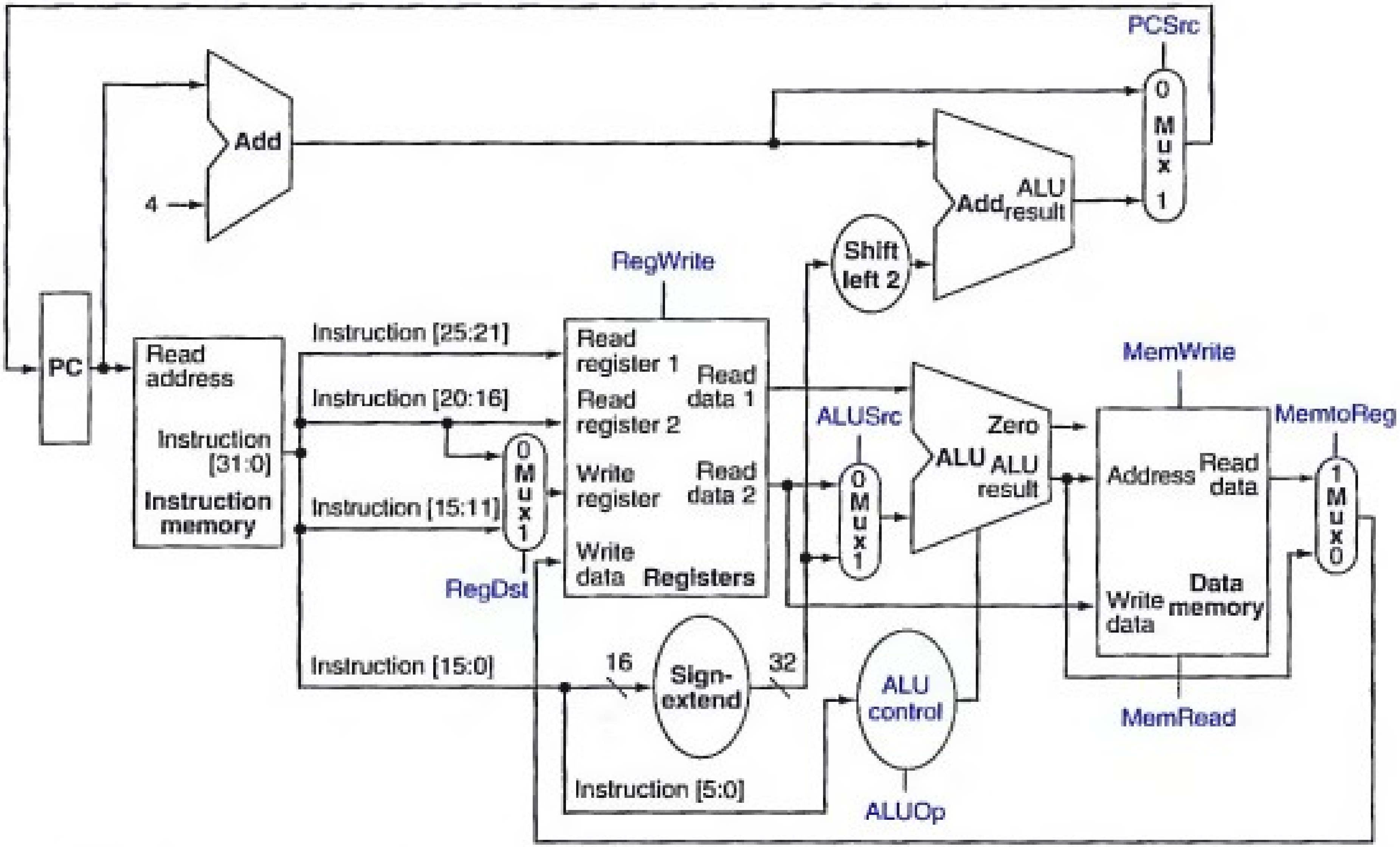
- opcode is always contained in bits 31:26
- The two registers to be read are always specified by the rs and rt fields at positions 25:21 and 20:16
  - r-type
  - branch equal
  - store
- The base register for load and store instruction is always in bit positions 25:21 i.e. rs
- The 16 bit offset for branch equal, load and store is always in positions 15:0
- The destination register is in one of two places.
  - load instruction →rt 20:16
  - r type instruction →rd 15:11

Execution steps: add $s1, $s2, $s3

1. The instruction is fetched from _______, and the PC is incremented to _________
2. Two registers ______ and _______ are read from the register file
3. The ALU operates on the data read from the register file, based on the _______code
4. The result from the ALU is written into the ________register in register file

1. IR, 4
2. $s2, $s3
3. Opcode
4. $s1

Name	Format	Example						Comments
add	R	0	18	19	17	0	32	add $s1, $s2, $s3


PCSrc
Add
4
0
M
u
x
1
Add
ALU result
Shift left 2
RegWrite
PC
Read address
Instruction [31:0]
Instruction memory
Instruction [25:21]
Instruction [20:16]
Instruction [15:11]
Instruction [15:0]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
RegDst
ALUSrc
ALU
Zero
ALU result
MemWrite
MemtoReg
Address
Read data
Write data
Data memory
MemRead
16
Sign-extend
32
ALU control
ALUOp
Instruction [5:0]

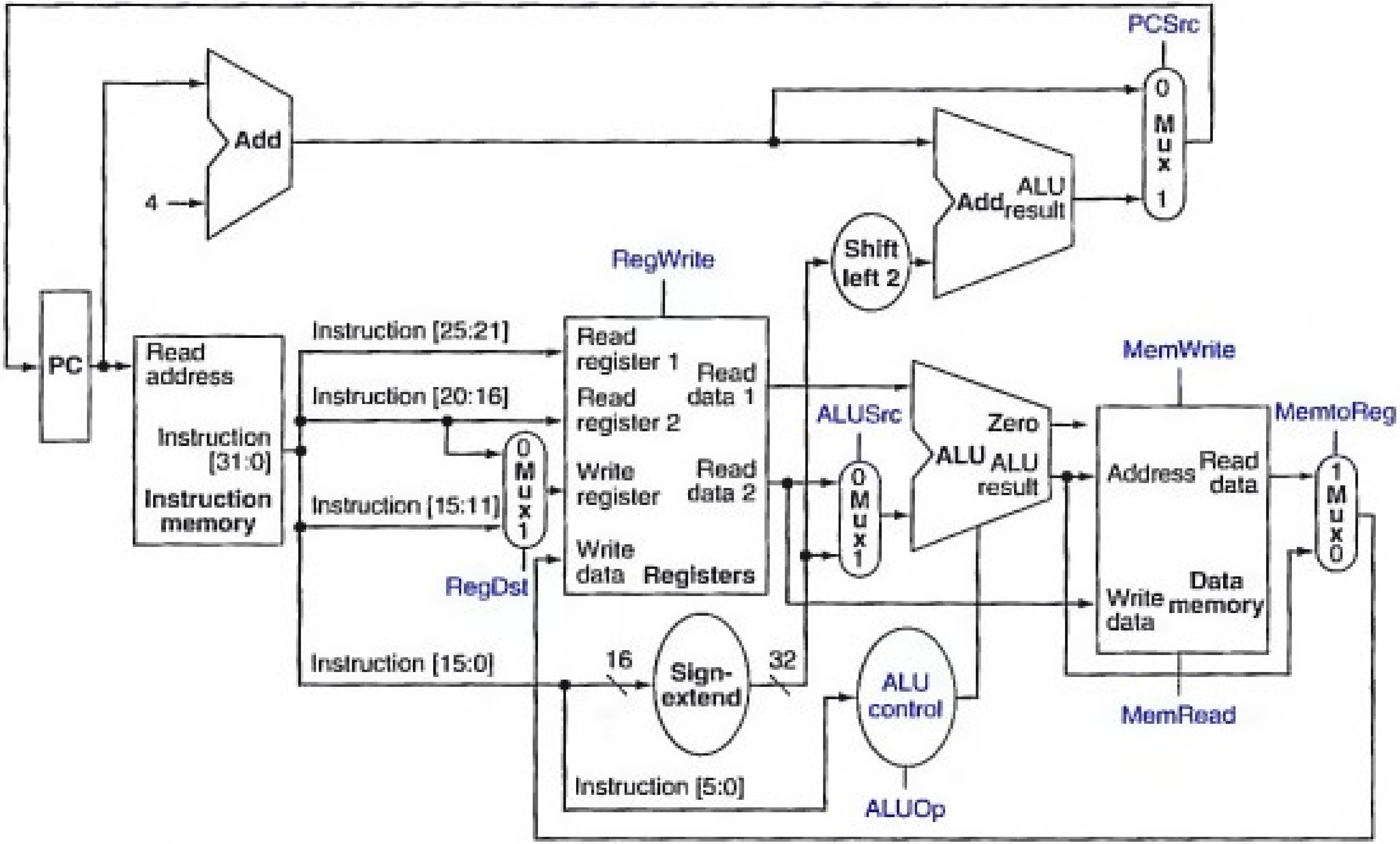
The datapath with all necessary multiplexors and all control lines identified.

Execution steps: lw $s1, 100($s2)

1. The instruction is fetched, and the PC is incremented
2. A register _________value is read from the register file
3. The ALU computes the sum of the value read from the register file and the sign extended lower 16 bits of the instruction
4. the sum from the ALU is used as the address for the data memory
5. The data from the memory unit is written into _____ in register file

1. Offset
2. $s1

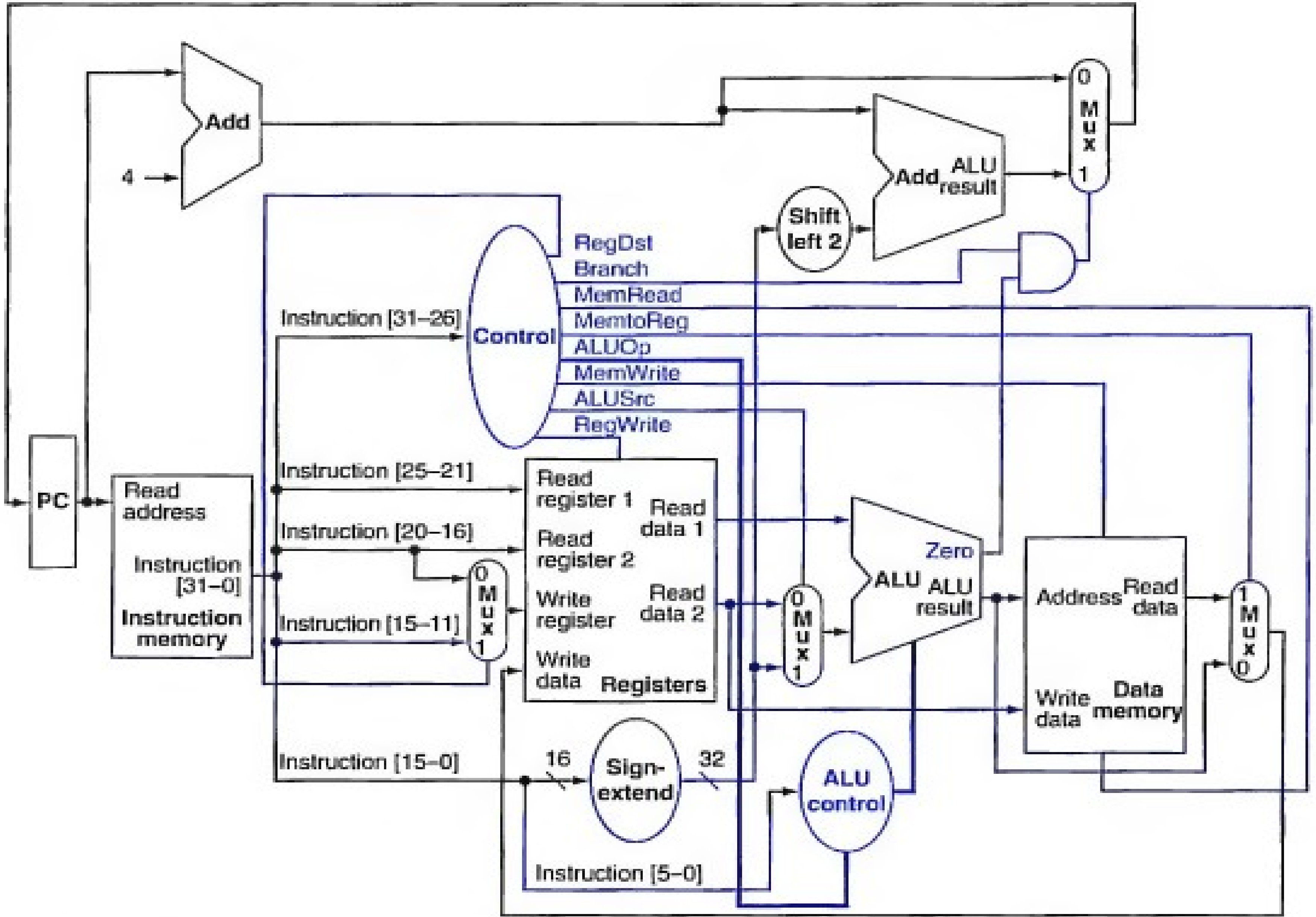
Name	Format	Example				Comments
lw	I	35	18	17	100	lw $s1, 100($s2)



PCSrc
Add
4
0
M
u
x
1
Add
ALU result
Shift left 2
RegWrite
PC
Read address
Instruction [31:0]
Instruction memory
Instruction [25:21]
Instruction [20:16]
Instruction [15:11]
Instruction [15:0]
Instruction [5:0]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
RegDst
16
32
Sign-extend
ALUSrc
ALU
Zero
ALU result
ALU control
ALUOp
MemWrite
Address
Read data
Write data
Data memory
MemRead
MemtoReg

The datapath with all necessary multiplexors and all control lines identified.

Complete Data Path


Add
4
ALU
Add result
Shift left 2
0
Mux
1
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Instruction [31–26]
Control
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
0
Mux
1
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
0
Mux
1
Zero
ALU
ALU result
Address
Read data
Write data
Data memory
1
Mux
0
Instruction [15–0]
16
Sign-extend
32
ALU control
Instruction [5–0]

THANK YOU

BITS Pilani
Pilani Campus

BITS Pilani
Pilani Campus

COMPUTING AND DESIGN

SESSION 6

Course design by - Dr. Lucy J. Gudino &
Dr Chandrasekhar
Faculty – Thangakumar J
WILP & Department of CS & IS

BITS Pilani
Pilani Campus

Operating systems – case study

Android …

- **Android** is a *Linux-based* platform for *mobile devices …*
  - *Operating System*
  - *Middleware*
  - *Applications*
  - *Software Development Kit* (**SDK**)
- Which kind of **mobile devices** … (examples)

SMARTPHONES

TABLETS

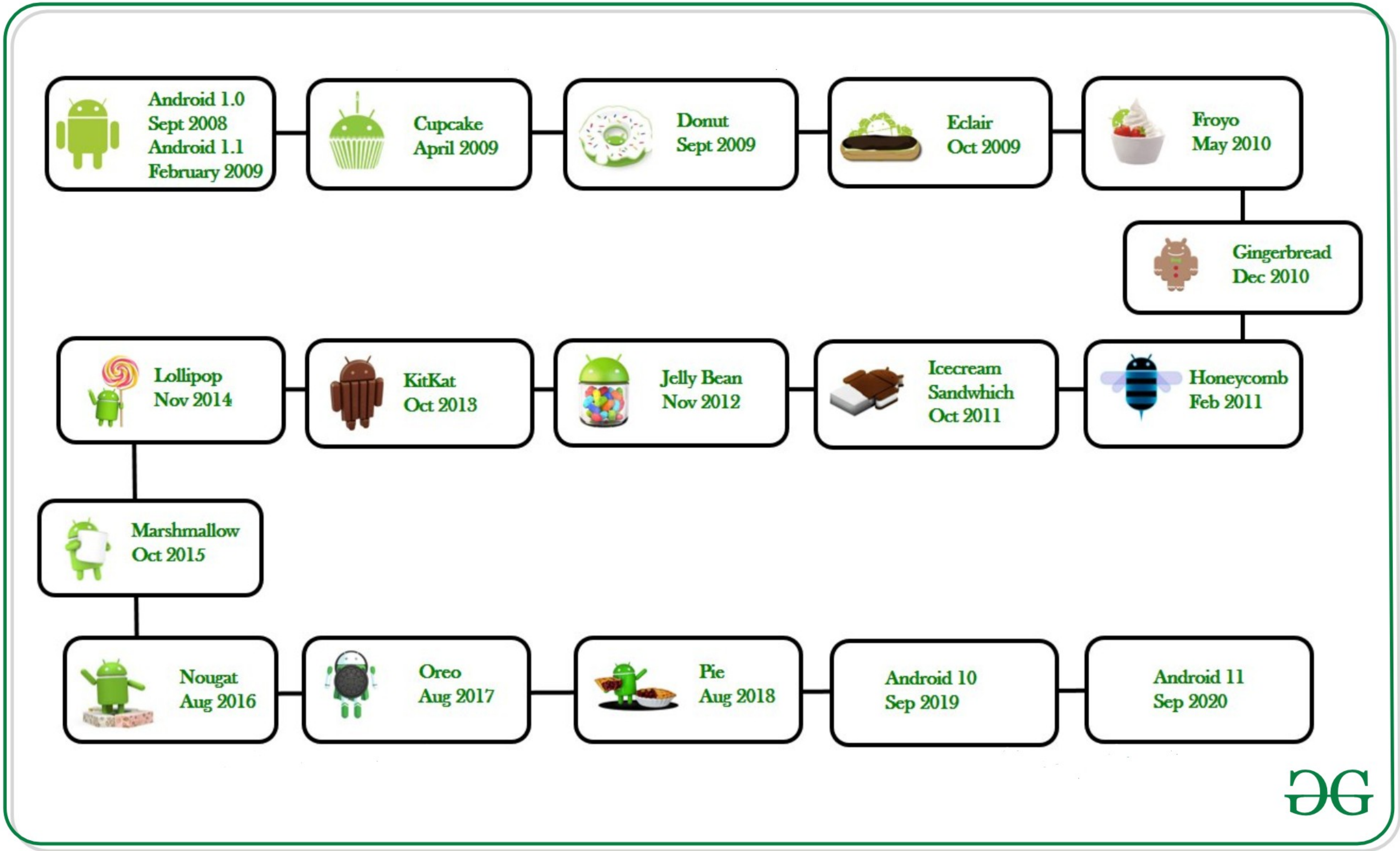
EREADERS

Android TV
HDTV and Internet Video on TV

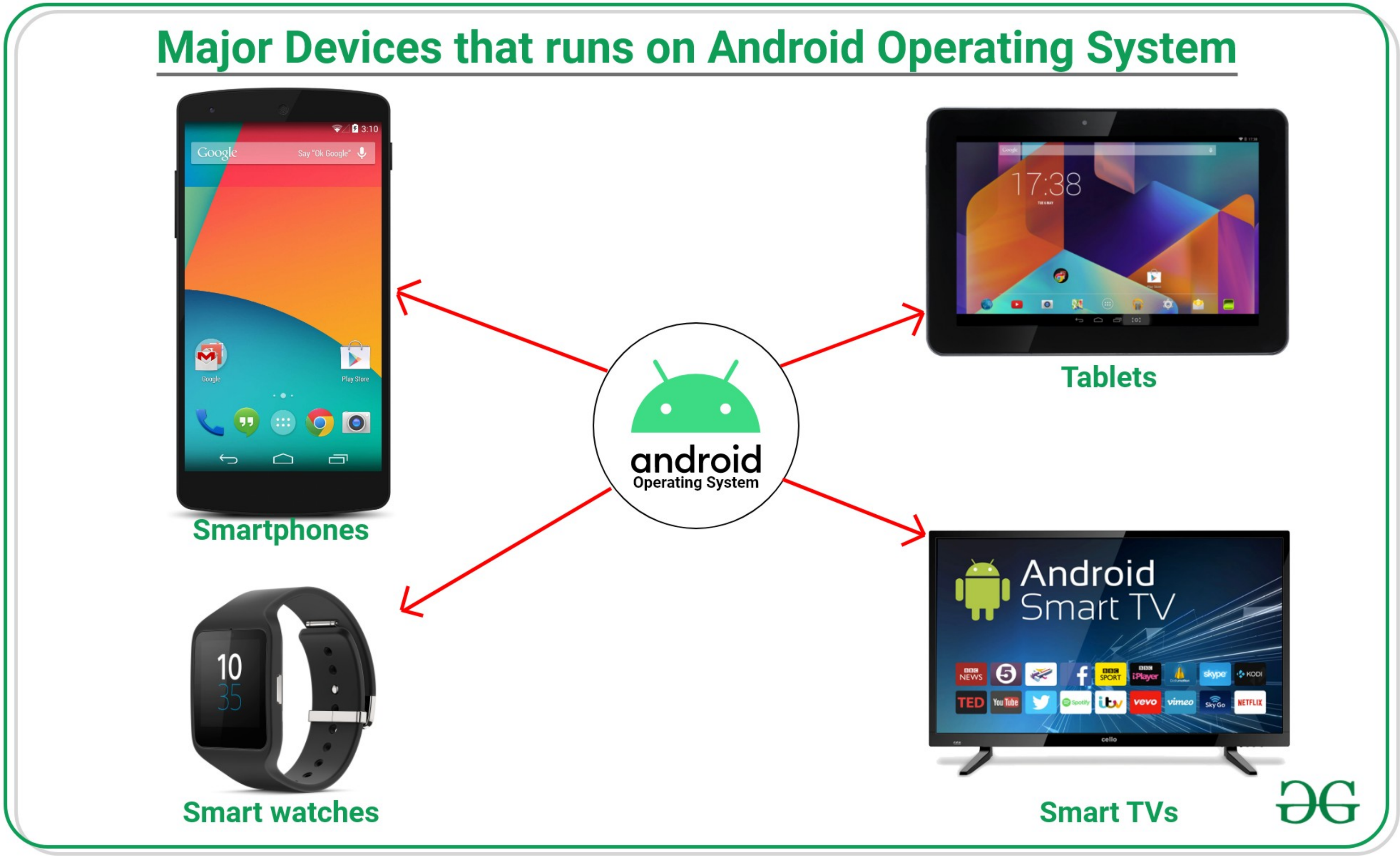
ANDROID TV

GOOGLE GLASSES

?

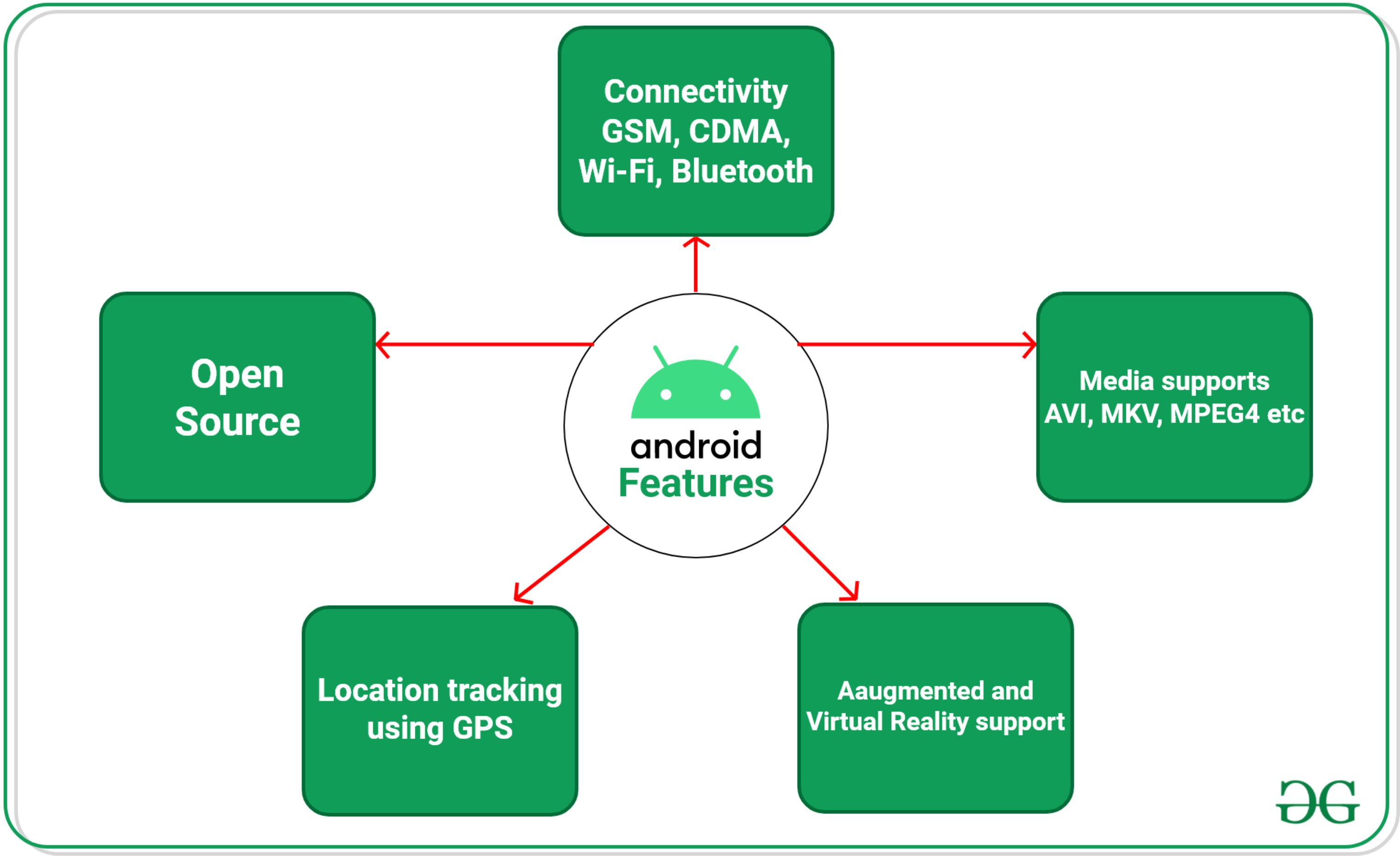

Android 1.0
Sept 2008
Android 1.1
February 2009
Cupcake
April 2009
Donut
Sept 2009
Eclair
Oct 2009
Froyo
May 2010
Gingerbread
Dec 2010
Honeycomb
Feb 2011
Icecream
Sandwhich
Oct 2011
Jelly Bean
Nov 2012
KitKat
Oct 2013
Lollipop
Nov 2014
Marshmallow
Oct 2015
Nougat
Aug 2016
Oreo
Aug 2017
Pie
Aug 2018
Android 10
Sep 2019
Android 11
Sep 2020

Versions of Android



Major Devices that runs on Android Operating System

android
Operating System
Smartphones
Tablets
Smart watches
Smart TVs
Android Smart TV



android

Features

Connectivity
GSM, CDMA,
Wi-Fi, Bluetooth

Open Source

Media supports
AVI, MKV, MPEG4 etc

Location tracking using GPS

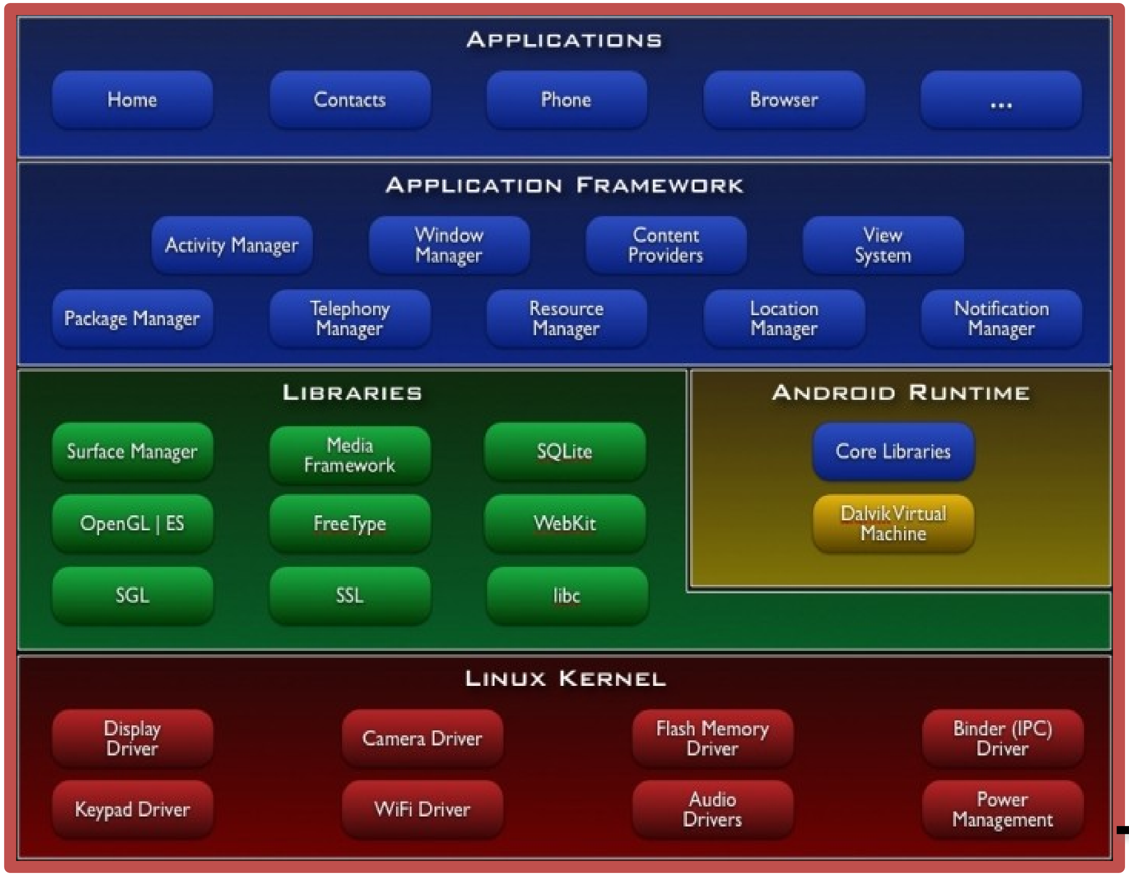
Aaugmented and
Virtual Reality support

- Android Open Source Project so we can customize the OS based on our requirements.
- Android supports different types of connectivity for GSM, CDMA, Wi-Fi, Bluetooth, etc. for telephonic conversation or data transfer.
- Using wifi technology we can pair with other devices while playing games or using other applications.
- It contains multiple APIs to support location-tracking services such as GPS.
- We can manage all data storage-related activities by using the file manager.

- It contains a wide range of media supports like AVI, MKV, FLV, MPEG4, etc. to play or record a variety of audio/video.
- It also supports different image formats like JPEG, PNG, GIF, BMP, MP3, etc.
- It supports multimedia hardware control to perform playback or recording using a camera and microphone.

- Android has an integrated open-source WebKit layout-based web browser to support User Interfaces like HTML5, and CSS3.
- Android supports multi-tasking means we can run multiple applications at a time and can switch between them.
- It provides support for virtual reality or 2D/3D Graphics.

The Android Architecture


Applications
Home
Contacts
Phone
Browser
...
Application Framework
Activity Manager
Window Manager
Content Providers
View System
Package Manager
Telephony Manager
Resource Manager
Location Manager
Notification Manager
Libraries
Surface Manager
Media Framework
SQLite
OpenGL | ES
FreeType
WebKit
SGL
SSL
libc
Android Runtime
Core Libraries
Dalvik Virtual Machine
Linux Kernel
Display Driver
Camera Driver
Flash Memory Driver
Binder (IPC) Driver
Keypad Driver
WiFi Driver
Audio Drivers
Power Management

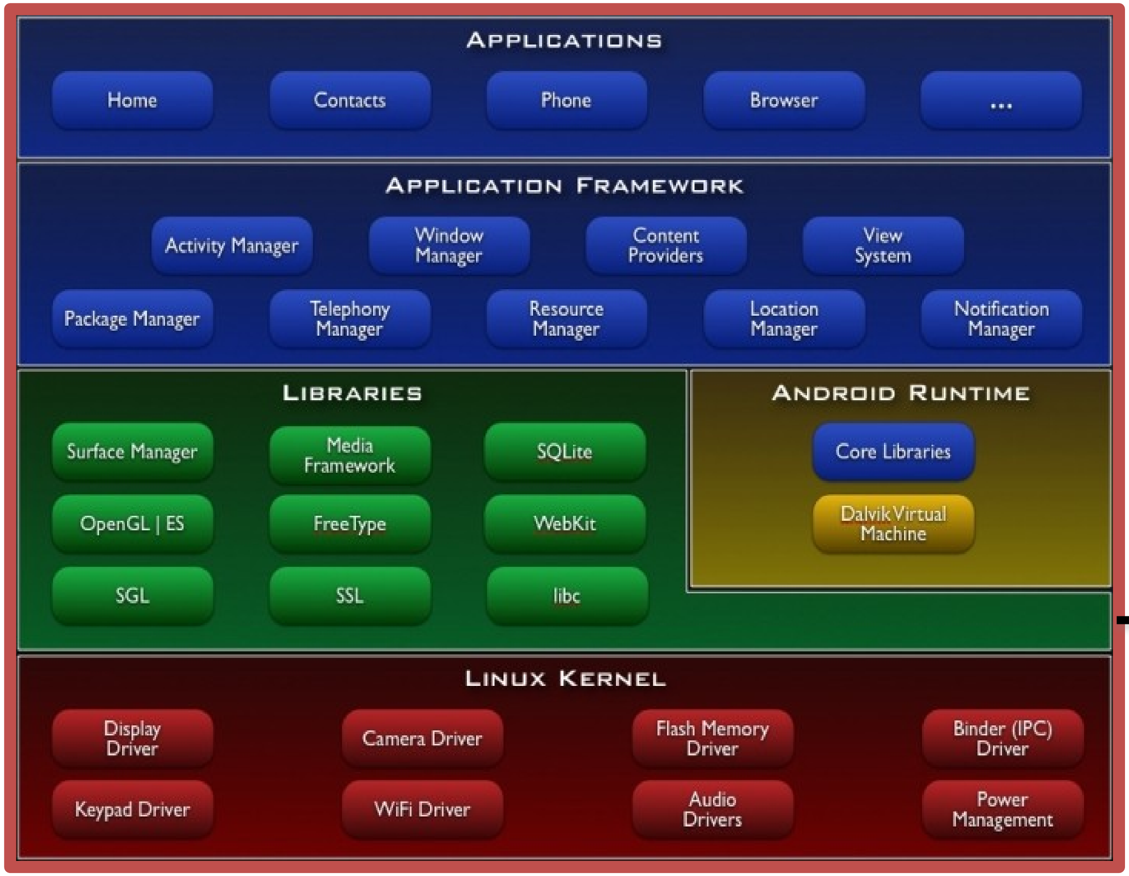
Built on top of
Linux kernel

Advantages:

- **Portability** (i.e. easy to compile on different harwdare architectures)
- **Security** (e.g. secure multi-process environment)
- **Power** Management

• Memory and Process management ▫ File & Network I/O ▫ Device Drivers • Preemptive Multitasking • Lean, efficient, and secure • Open Source

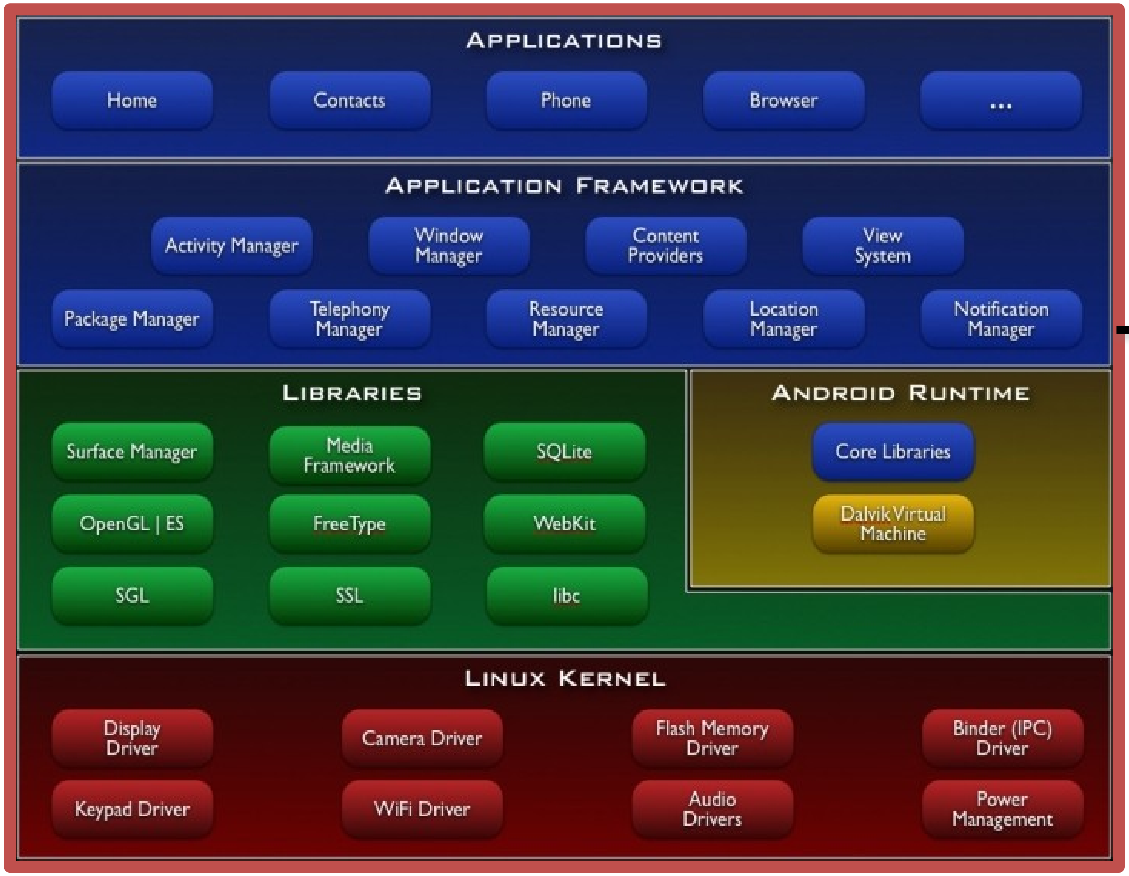
The Android Architecture


APPLICATIONS
Home
Contacts
Phone
Browser
...
APPLICATION FRAMEWORK
Activity Manager
Window Manager
Content Providers
View System
Package Manager
Telephony Manager
Resource Manager
Location Manager
Notification Manager
LIBRARIES
Surface Manager
Media Framework
SQLite
OpenGL | ES
FreeType
WebKit
SGL
SSL
libc
ANDROID RUNTIME
Core Libraries
Dalvik Virtual Machine
LINUX KERNEL
Display Driver
Camera Driver
Flash Memory Driver
Binder (IPC) Driver
Keypad Driver
WiFi Driver
Audio Drivers
Power Management

Native **Libraries**
(C/C++ code)

- **Graphics** (Surface Manager)
- **Multimedia** (Media Framework)
- **Database DBMS** (SQLite)
- **Font Management** (FreeType)
- **WebKit**
- **C libraries** (Bionic)
- **....**

The Android Architecture

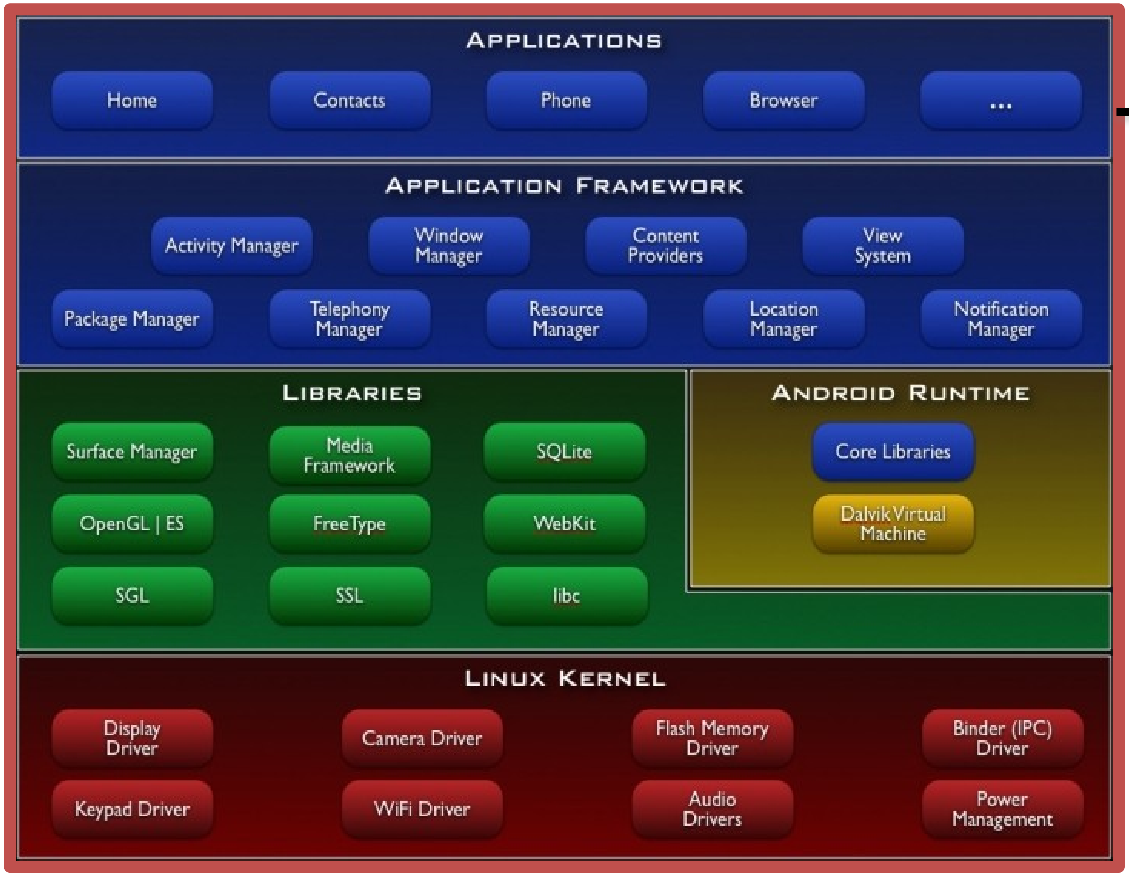

APPLICATIONS
Home
Contacts
Phone
Browser
...
APPLICATION FRAMEWORK
Activity Manager
Window Manager
Content Providers
View System
Package Manager
Telephony Manager
Resource Manager
Location Manager
Notification Manager
LIBRARIES
Surface Manager
Media Framework
SQLite
OpenGL | ES
FreeType
WebKit
SGL
SSL
libc
ANDROID RUNTIME
Core Libraries
Dalvik Virtual Machine
LINUX KERNEL
Display Driver
Camera Driver
Flash Memory Driver
Binder (IPC) Driver
Keypad Driver
WiFi Driver
Audio Drivers
Power Management

Application Libraries

(Core Components of Android)

- **Activity Manager**
- **Packet Manager**
- **Telephony Manager**
- **Location Manager**
- **Contents Provider**
- **Notification Manager**
- **….**

The Android Architecture

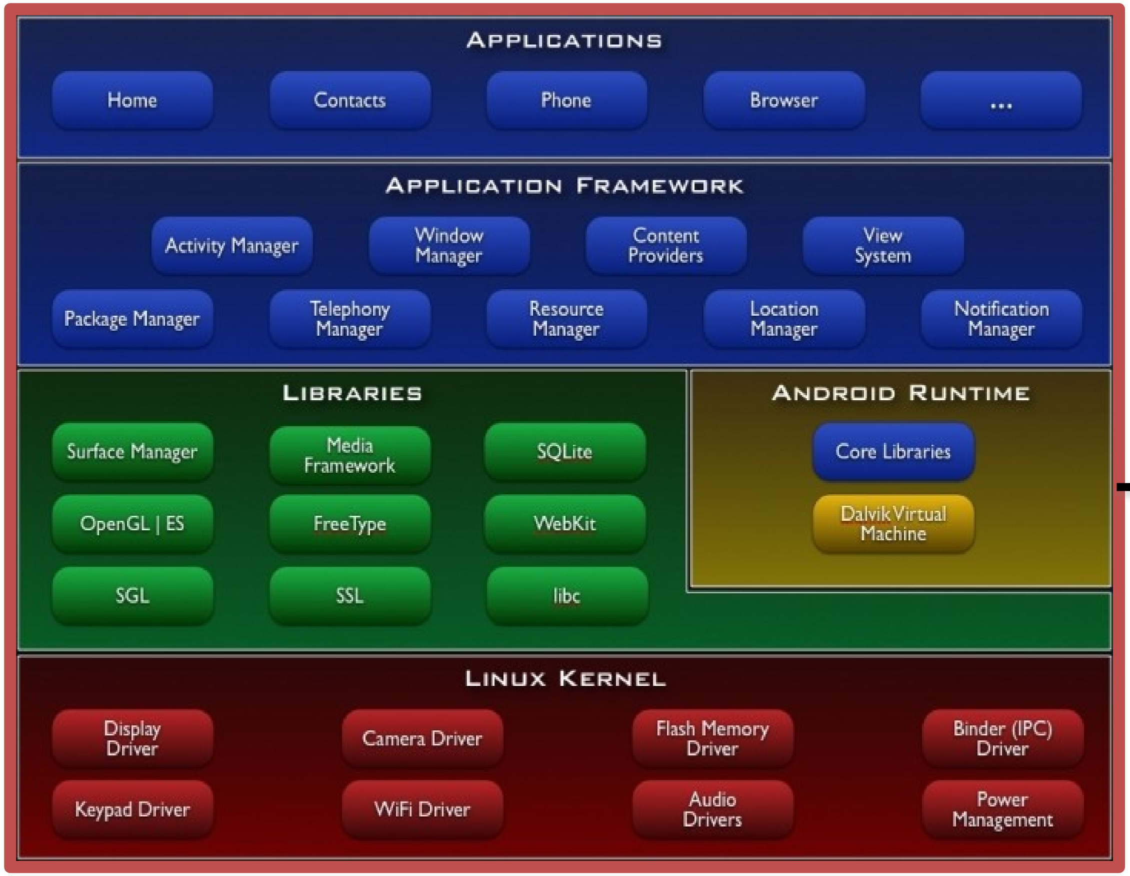

APPLICATIONS
Home
Contacts
Phone
Browser
...
APPLICATION FRAMEWORK
Activity Manager
Window Manager
Content Providers
View System
Package Manager
Telephony Manager
Resource Manager
Location Manager
Notification Manager
LIBRARIES
Surface Manager
Media Framework
SQLite
OpenGL | ES
FreeType
WebKit
SGL
SSL
libc
ANDROID RUNTIME
Core Libraries
Dalvik Virtual Machine
LINUX KERNEL
Display Driver
Camera Driver
Flash Memory Driver
Binder (IPC) Driver
Keypad Driver
WiFi Driver
Audio Drivers
Power Management

Applications

(Written in **Java** code)

- **Android Play Store**
- **Entertainment**
- **Productivity**
- **Personalization**
- **Education**
- **Geo-communication**
- **….**

The Android Architecture


APPLICATIONS
Home
Contacts
Phone
Browser
...
APPLICATION FRAMEWORK
Activity Manager
Window Manager
Content Providers
View System
Package Manager
Telephony Manager
Resource Manager
Location Manager
Notification Manager
LIBRARIES
Surface Manager
Media Framework
SQLite
OpenGL | ES
FreeType
WebKit
SGL
SSL
libc
ANDROID RUNTIME
Core Libraries
Dalvik Virtual Machine
LINUX KERNEL
Display Driver
Camera Driver
Flash Memory Driver
Binder (IPC) Driver
Keypad Driver
WiFi Driver
Audio Drivers
Power Management

Dalvik Virtual Machine (VM)

- **Novel** Java Virtual Machine implementation (not using the Oracle JVM)
- Open **License** (Oracle JVM is not open!)
- **Optimized** for memory-constrained devices
- **Faster** than Oracle JVM
- ….

Android Applications Design

APPLICATION **DESIGN**:

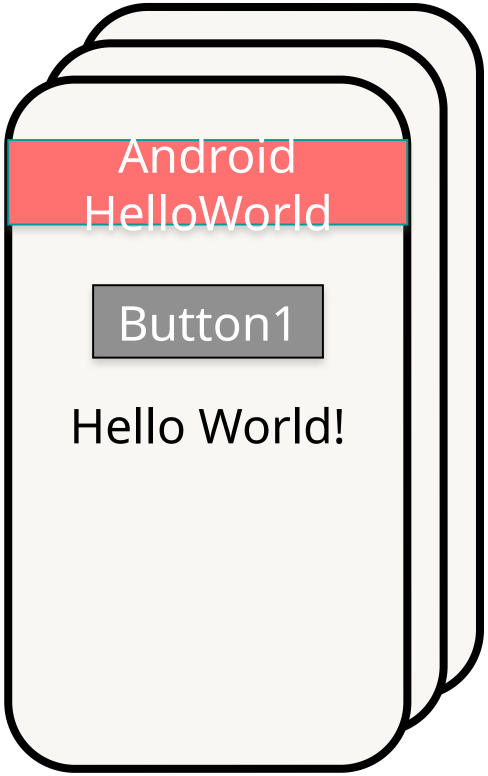
- **GUI** Definition
- **Events** Management
- Application **Data** Management
- **Background** Operations
- **User** Notifications

Android Applications Design

APPLICATION **COMPONENTS**

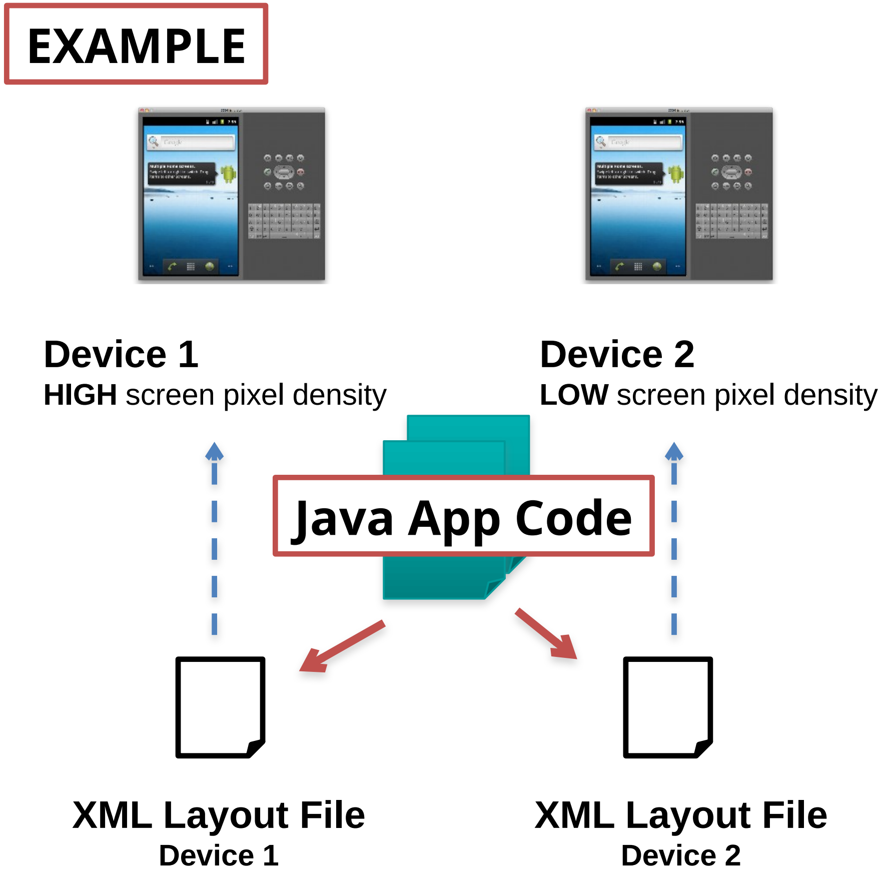
- **Activities**
- **Intents**
- **Services**
- **Content Providers**
- **Broadcast Receivers**

Android Components: Activities


Android HelloWorld
Button1
Hello World!

- An **Activity** corresponds to a **single screen** of the **Application**.
- An Application can be composed of *multiples screens* (Activities).
- The **Home Activity** is shown when the user launches an application.
- Different activities can exhange information one with each other.

Android Components: Activities



EXAMPLE

Device 1
HIGH screen pixel density

Device 2
LOW screen pixel density

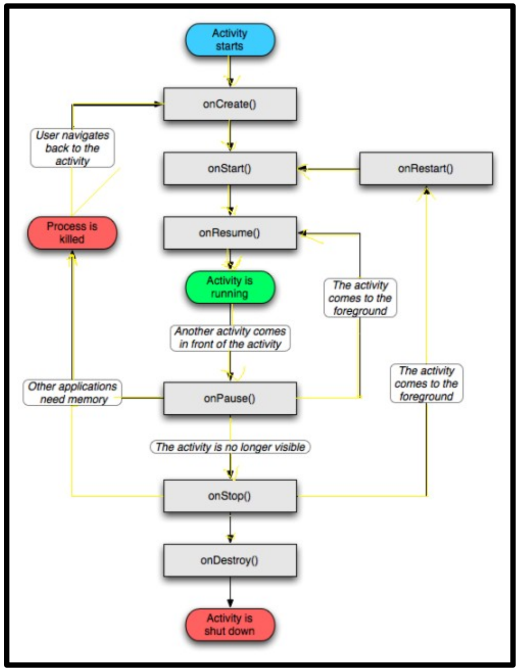
Java App Code

XML Layout File
Device 1

XML Layout File
Device 2

- Build the **application layout** through XML files (like HTML)
- Define **two** different XML **layouts** for two different devices
- At **runtime**, Android detects the current device configuration and loads the appropriate resources for the application
- **No need to recompile**!
- Just add a new XML file if you need to support a new device

Android Components: Activities


Activity starts
onCreate()
User navigates back to the activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
The activity comes to the foreground
Another activity comes in front of the activity
Other applications need memory
onPause()
The activity comes to the foreground
The activity is no longer visible
onStop()
onDestroy()
Activity is shut down

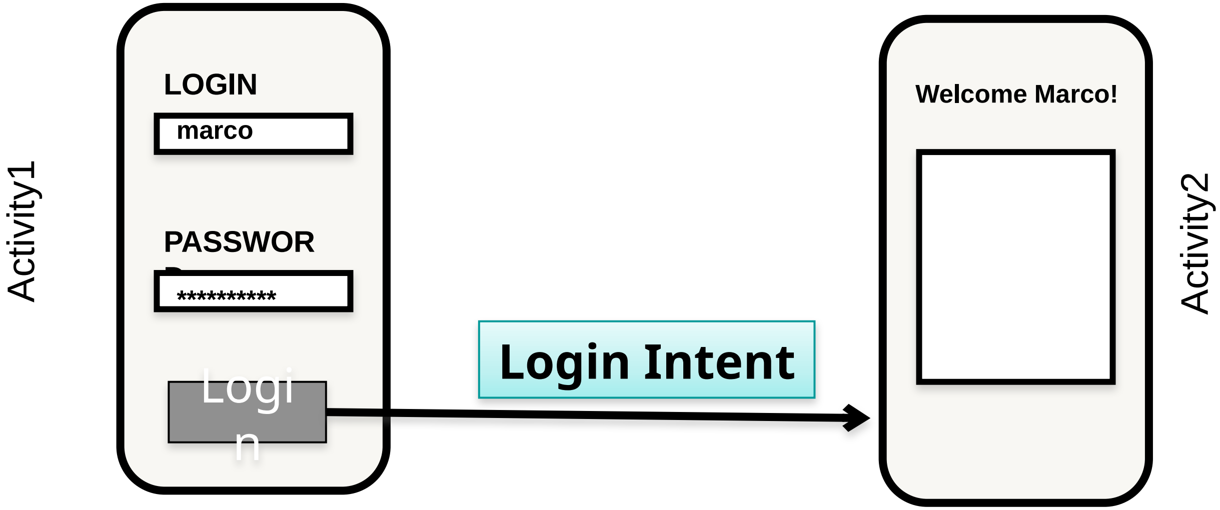
- The **Activity Manager** is responsible for creating, destroying, managing activities.
- Activities can be on different **states**: *starting*, *running*, *stopped*, *destroyed*, *paused*.
- Only one activity can be on the **running** state at a time.
- Activities are organized on a **stack**, and have an event-driven life cycle (details later …)

Android Components: Activities

- Main difference between Android-programming and Java (Oracle) -programming:
  - **Mobile devices have constrained resource capabilities!**
- Activity lifetime depends on **users' choice** (i.e. change of visibility) as well as on **system contraints** (i.e. memory shortage).
- Developer must implement **lifecycle methods** to account for state changes of each Activity …

Android Components: Intents

- **Intents**: asynchronous **messages** to activate core Android components (e.g. Activities).
- **Explicit** Intent → The component *(e.g. Activity1)* specifies the destination of the intent *(e.g. Activity 2)*.



Activity1

LOGIN

marco

PASSWORD

****

Login

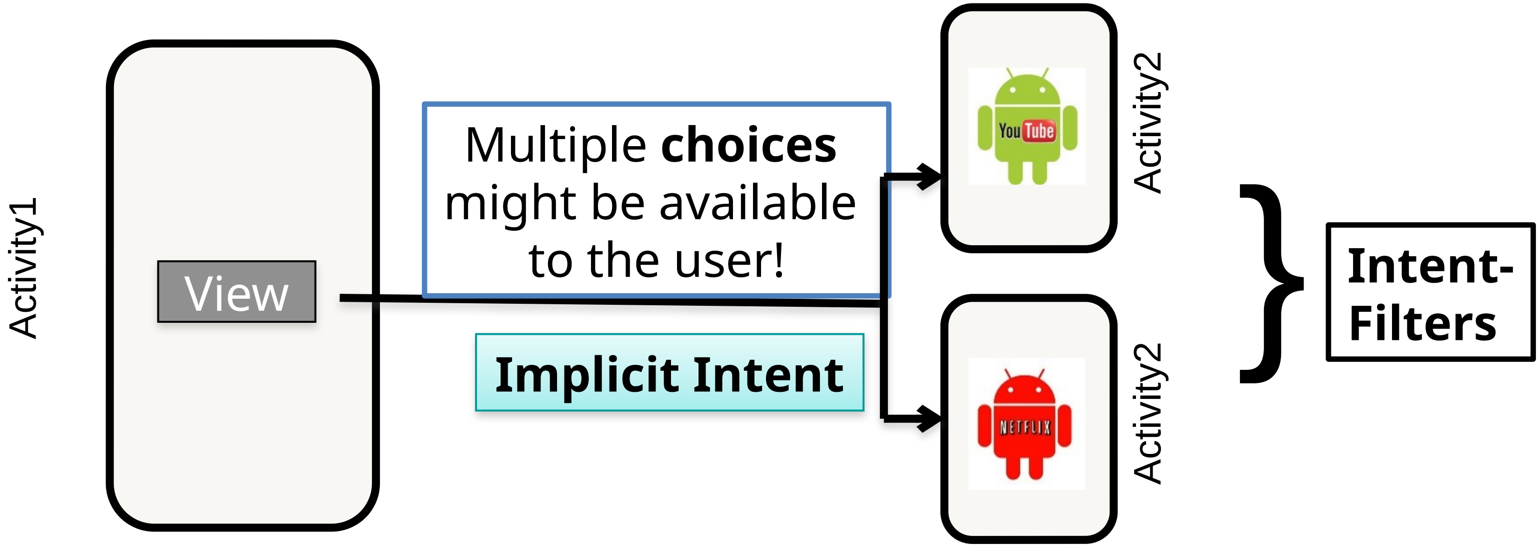
Login Intent

Welcome Marco!

Activity2

Android Components: Intents

- **Intents**: asynchronous **messages** to activate core Android components (e.g. Activities).
- **Implicit** Intent → The component *(e.g. Activity1)* specifies the type of the intent *(e.g. "View a video")*.


Activity1
View
Activity2
Activity2
You Tube
NETFLIX

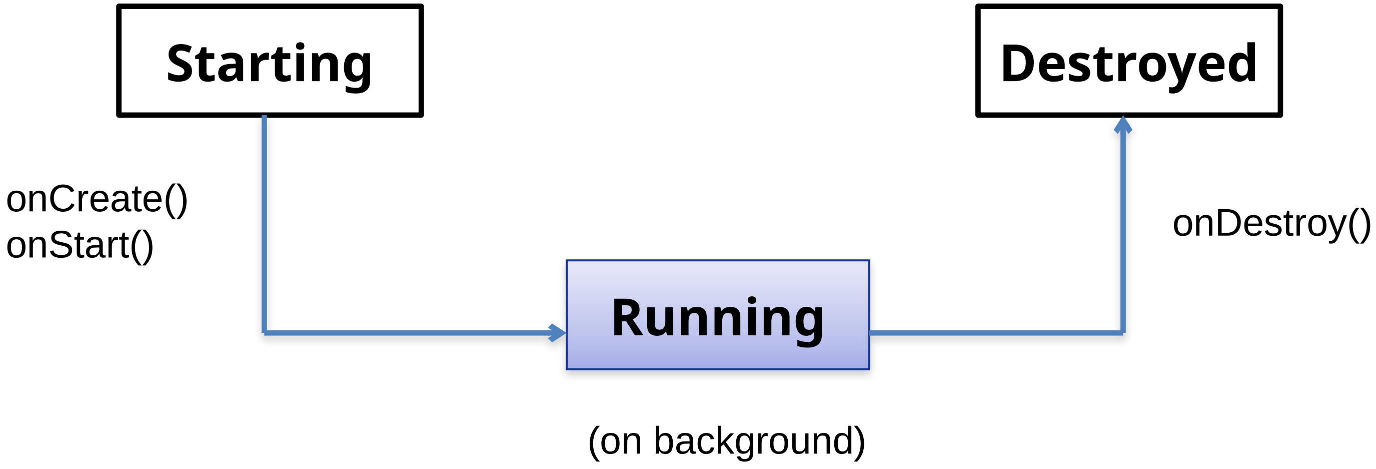
Multiple **choices** might be available to the user!

Implicit Intent

Intent-Filters

Android Components: Services

- **Services**: like Activities, but run in **background** and do not provide an user interface.
- Used for **non-interactive** tasks (e.g. networking).
- Service life-time composed of 3 states:



Starting

Destroyed

onCreate()
onStart()

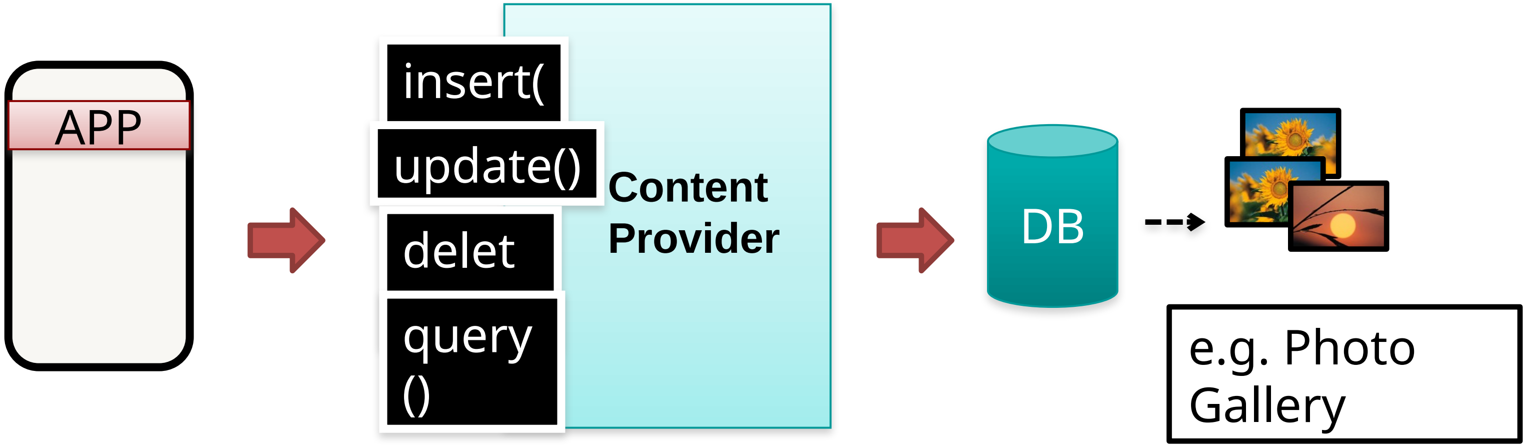
onDestroy()

Running

(on background)

Android Components: Content Providers

- Each Android **application** has its own **private** set of data (managed through *files* or through *SQLite* database).
- **Content Providers**: Standard **interface** to *access and share data among different applications*.



APP

insert(

update()

delet

query ()

Content Provider

DB

e.g. Photo Gallery

Android Components: System API

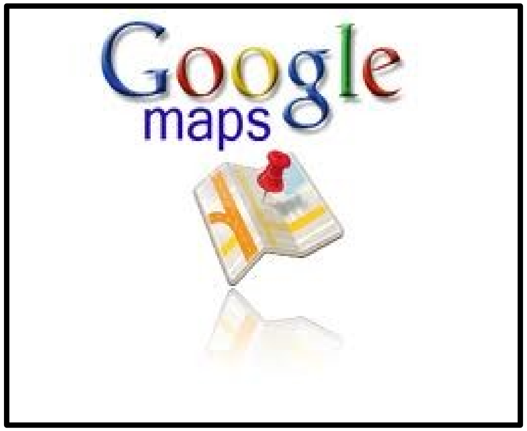
- Using the **components** described so far, Android applications can then leverage the system API …

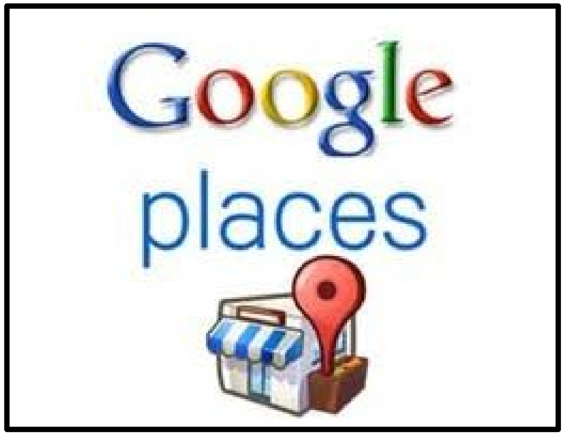
SOME EXAMPLEs …

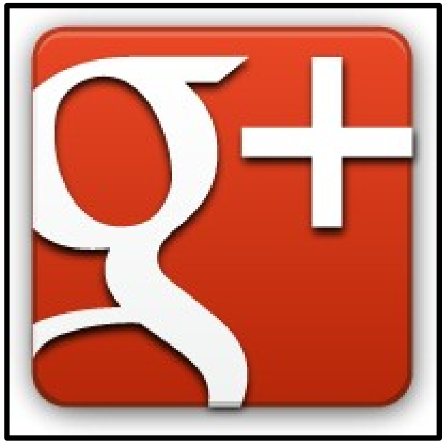
  - *Telephony Manager* data access (call, SMS, etc)
  - *Sensor* management (GPS, accelerometer, etc)
  - Network *connectivity* (Wifi, bluetooth, NFC, etc)
  - *Web* surfing (HTTP client, WebView, etc)
  - *Storage* management (files, SQLite db, etc)
  - ….

Android Components: Google API

- … or easily interface with other **Google services**:


Google
maps


Google
places

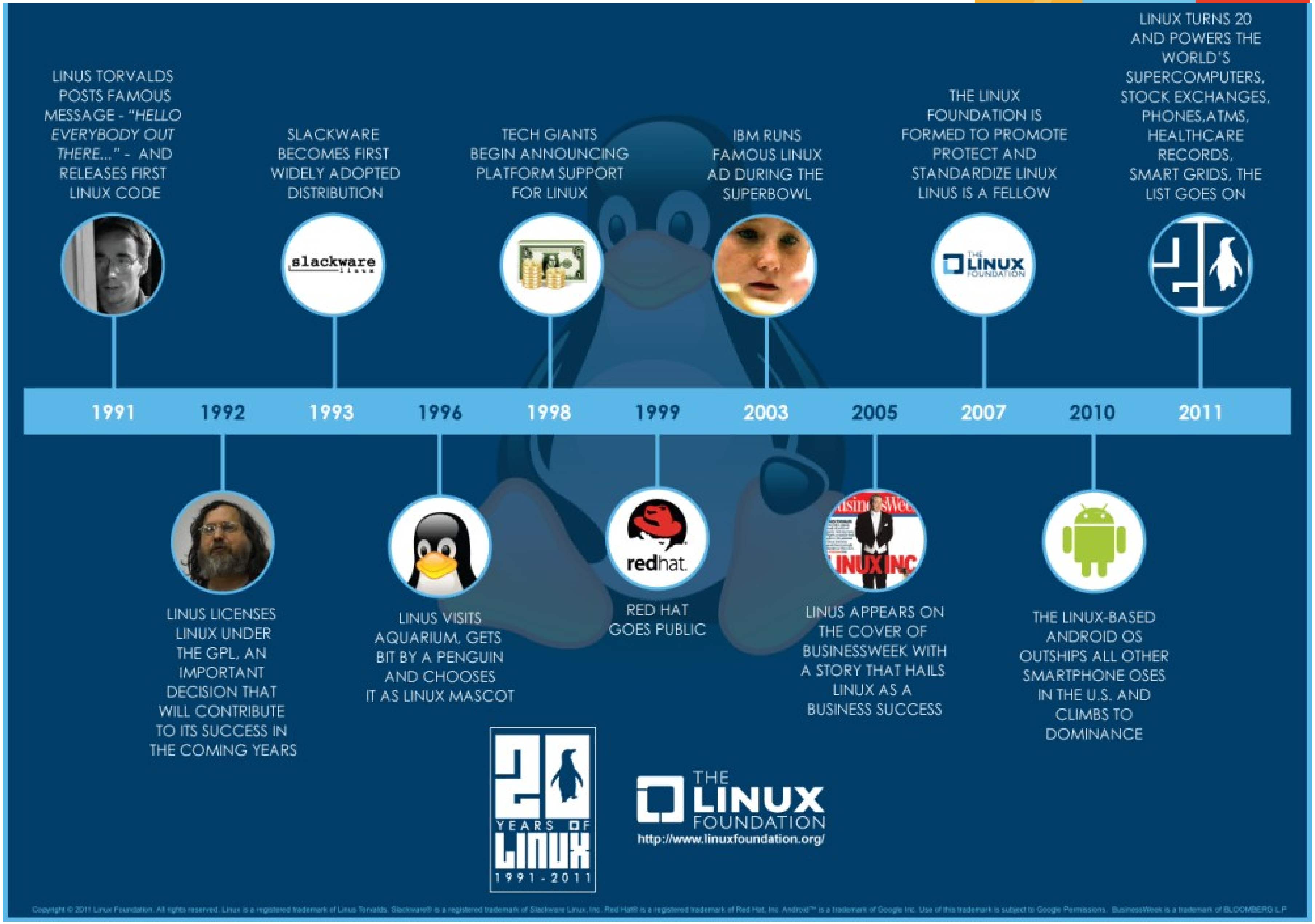


Android Application Security

- Android applications run with a distinct system identity (Linux user ID and group ID), in an **isolated** way.
- Applications must explicitly share resources and data. They do this by declaring the ***permissions*** they need for additional capabilities.
  - Applications statically **declare** the permissions they require.
  - User must **give his/her consensus** during the installation.

Case Study - Linux

1. **Evolution of UNIX**
2. **LINUX Operating System & Design goals**
3. **Interfaces to Linux**
4. **List of Linux Utility Programs**
5. **Linux kernel**
6. **Process in Linux and PID, UID, GID in Linux**
7. **Process Management System Calls in Linux**
8. **User space thread and kernel space thread**
9. **Booting process in Linux**
10. **Memory management in Linux**
11. **Paging in Linux**
12. **Network File System calls in Linux**



1991
LINUS TORVALDS POSTS FAMOUS MESSAGE - *"HELLO EVERYBODY OUT THERE..."* - AND RELEASES FIRST LINUX CODE

1992
LINUS LICENSES LINUX UNDER THE GPL, AN IMPORTANT DECISION THAT WILL CONTRIBUTE TO ITS SUCCESS IN THE COMING YEARS

1993
SLACKWARE BECOMES FIRST WIDELY ADOPTED DISTRIBUTION

1996
LINUS VISITS AQUARIUM, GETS BIT BY A PENGUIN AND CHOOSES IT AS LINUX MASCOT

1998
TECH GIANTS BEGIN ANNOUNCING PLATFORM SUPPORT FOR LINUX

1999
RED HAT GOES PUBLIC

2003
IBM RUNS FAMOUS LINUX AD DURING THE SUPERBOWL

2005
LINUS APPEARS ON THE COVER OF BUSINESSWEEK WITH A STORY THAT HAILS LINUX AS A BUSINESS SUCCESS

2007
THE LINUX FOUNDATION IS FORMED TO PROMOTE PROTECT AND STANDARDIZE LINUX LINUS IS A FELLOW

2010
THE LINUX-BASED ANDROID OS OUTSHIPS ALL OTHER SMARTPHONE OSES IN THE U.S. AND CLIMBS TO DOMINANCE

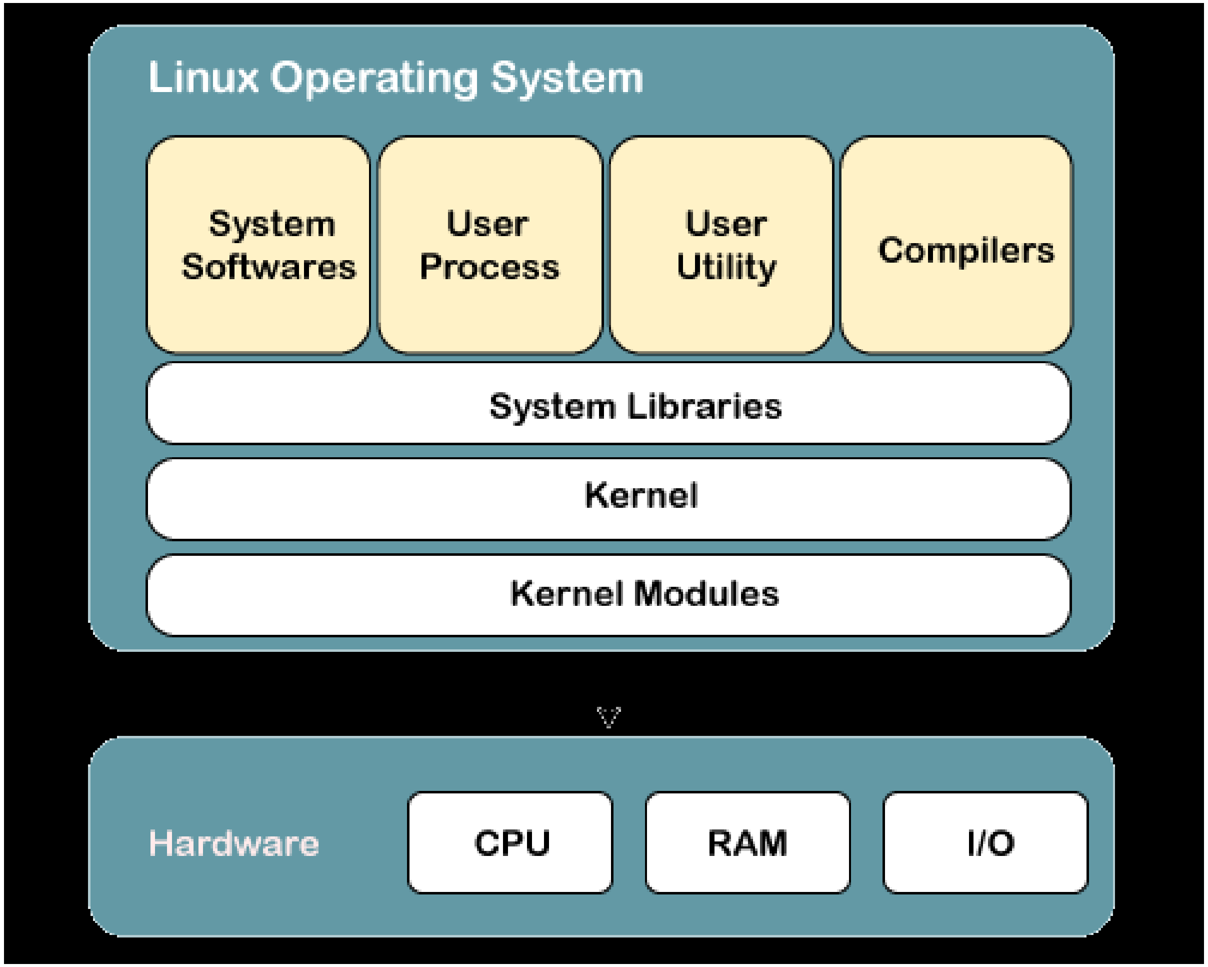
2011
LINUX TURNS 20 AND POWERS THE WORLD'S SUPERCOMPUTERS, STOCK EXCHANGES, PHONES,ATMS, HEALTHCARE RECORDS, SMART GRIDS, THE LIST GOES ON

20 YEARS OF LINUX
1991-2011

THE LINUX FOUNDATION
http://www.linuxfoundation.org/

Copyright © 2011 Linux Foundation. All rights reserved. Linux is a registered trademark of Linus Torvalds. Slackware® is a registered trademark of Slackware Linux, Inc. Red Hat® is a registered trademark of Red Hat, Inc. Android™ is a trademark of Google Inc. Use of this trademark is subject to Google Permissions. BusinessWeek is a trademark of BLOOMBERG L.P.

Linux architecture

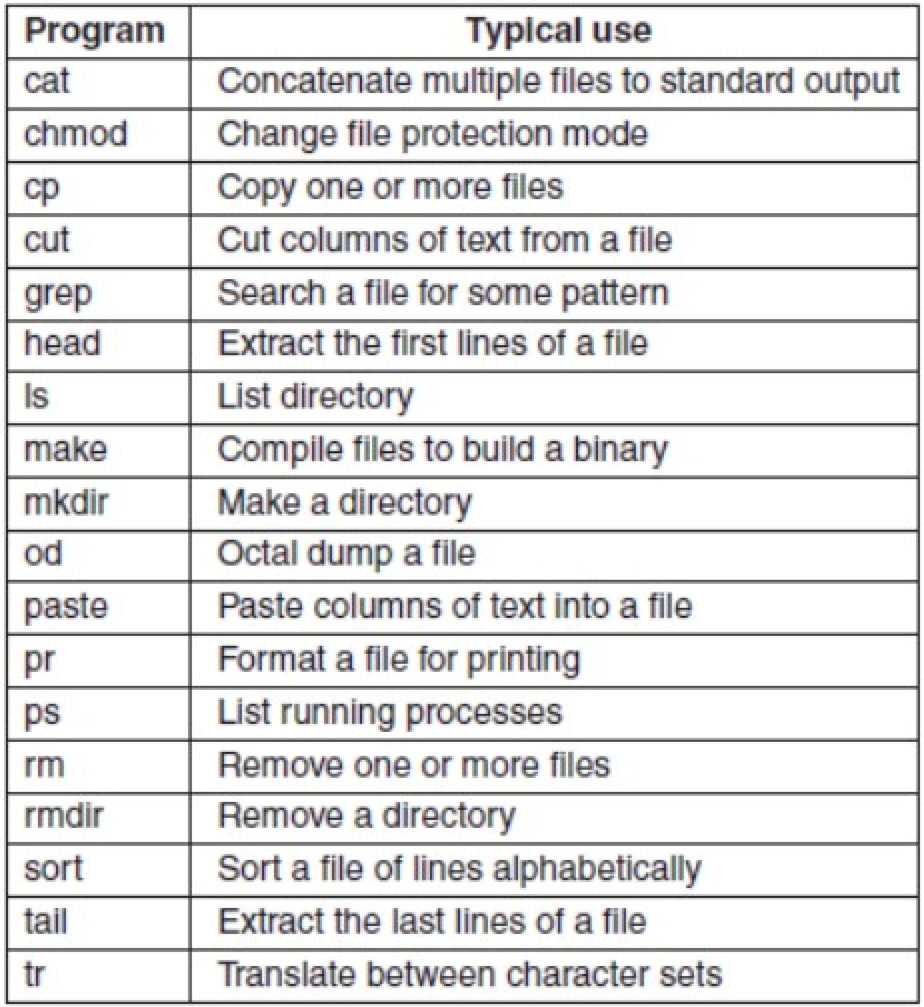


Linux Operating System
System Softwares
User Process
User Utility
Compilers
System Libraries
Kernel
Kernel Modules
Hardware
CPU
RAM
I/O

Design goals of Linux

- Designed to handle multiple processes and multiple users at the same time
- Multi-user, multitasking system with a full set of UNIX-compatible tools.
- Speed, efficiency, and standardization.
- Designed to be compliant with the relevant POSIX documents.
- The Linux programming interface adheres to the SVR4 UNIX semantics.

List of Linux Utility Programs

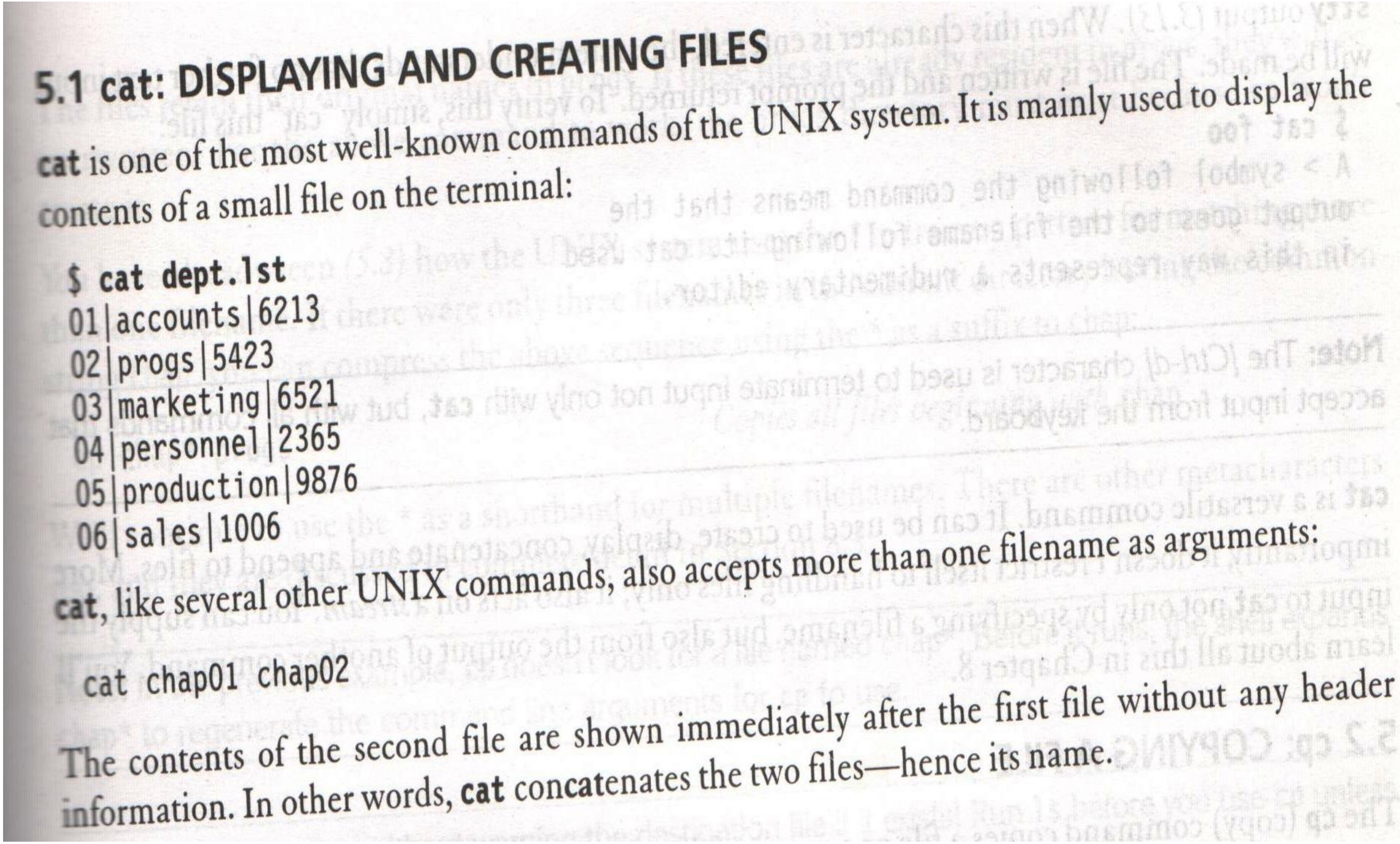


Program	Typical use
cat	Concatenate multiple files to standard output
chmod	Change file protection mode
cp	Copy one or more files
cut	Cut columns of text from a file
grep	Search a file for some pattern
head	Extract the first lines of a file
ls	List directory
make	Compile files to build a binary
mkdir	Make a directory
od	Octal dump a file
paste	Paste columns of text into a file
pr	Format a file for printing
ps	List running processes
rm	Remove one or more files
rmdir	Remove a directory
sort	Sort a file of lines alphabetically
tail	Extract the last lines of a file
tr	Translate between character sets

- Commands :

- **cat - **to create, view, concatenate files**.**
- **mv** - mv is used to move one or more files or directories
- **rm** - removes the entries for a specified file, group of files.
- **cmp, comm, diff** - to display the differences by comparing the files
- **chmod**- used to change the access permissions
- **top** - Displays list of running processes
- **head**- Pick lines from the beginning
- **tail** - Pick lines from the end
- **sort** - Sort, merge and remove
- **wc** - to find number of lines, word count, byte and characters.

cat – DISPLAYING AND CREATING FILES



5.1 cat: DISPLAYING AND CREATING FILES

cat is one of the most well-known commands of the UNIX system. It is mainly used to display the contents of a small file on the terminal:

```
$ cat dept.lst
01|accounts|6213
02|progs|5423
03|marketing|6521
04|personnel|2365
05|production|9876
06|sales|1006
```

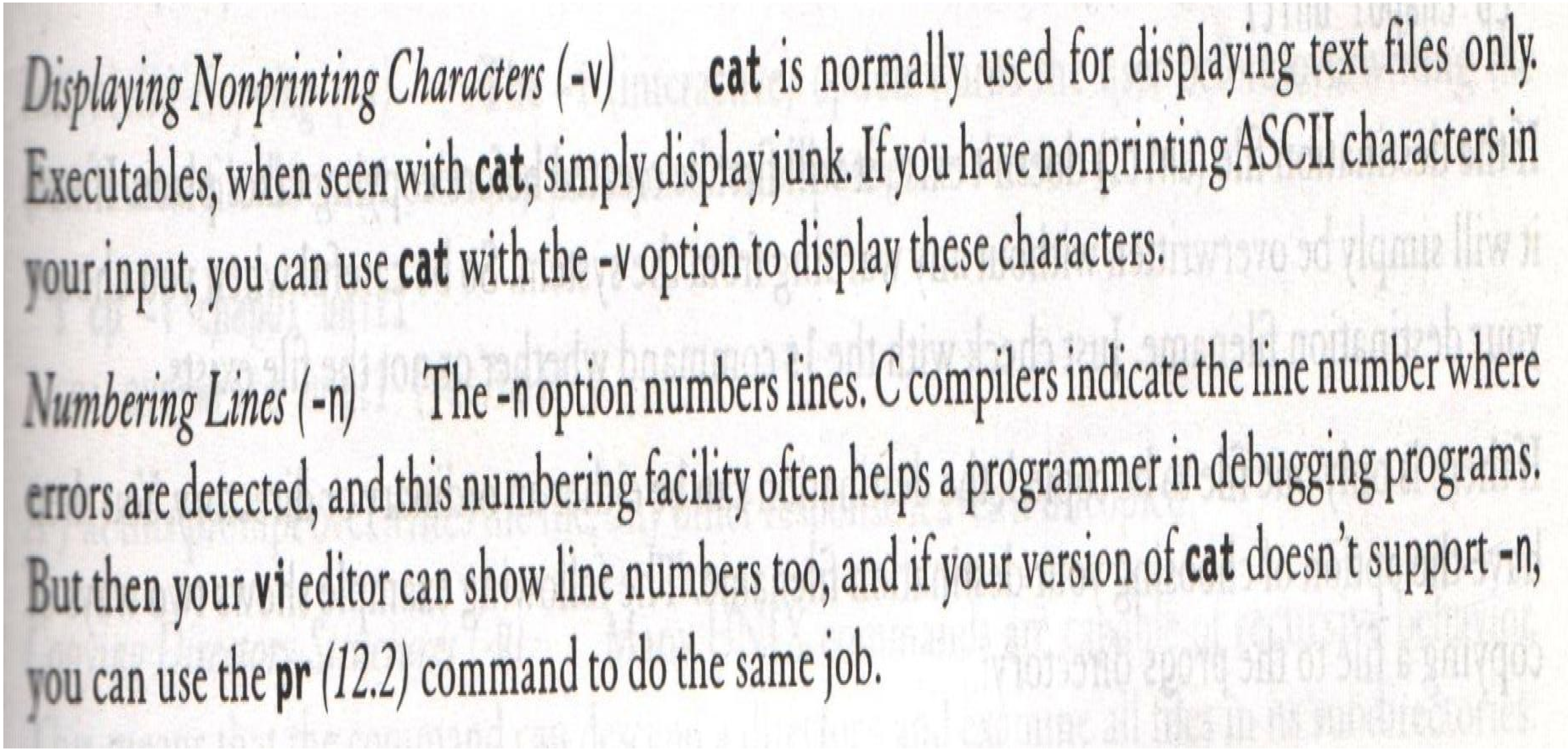
cat, like several other UNIX commands, also accepts more than one filename as arguments:

```
cat chap01 chap02
```

The contents of the second file are shown immediately after the first file without any header information. In other words, **cat** concatenates the two files—hence its name.

cat – DISPLAYING AND CREATING FILES

cat options (-v and –n)

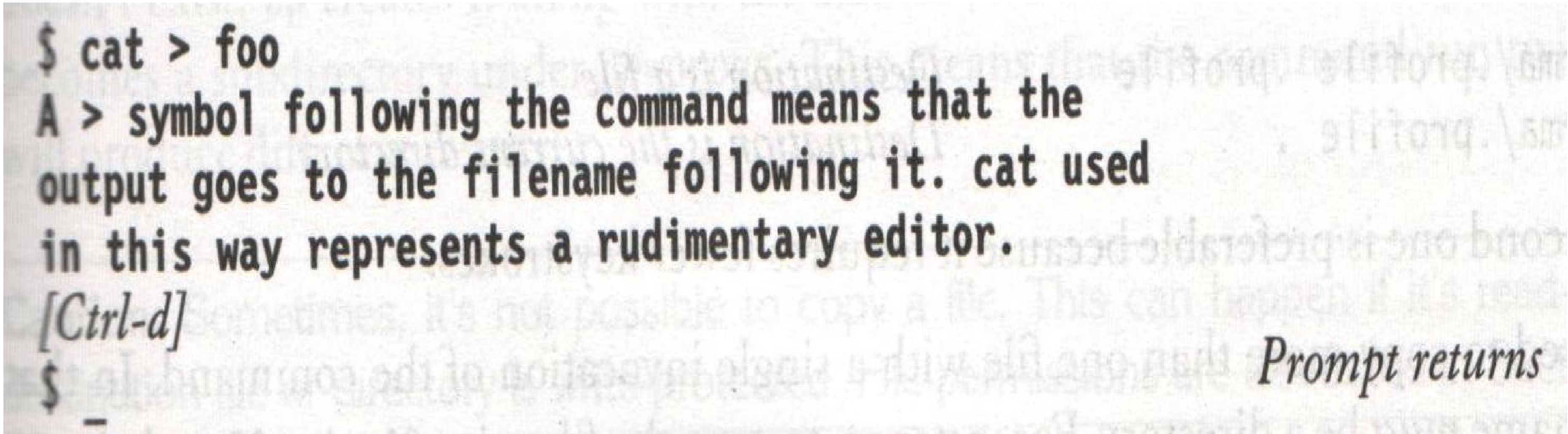


Displaying Nonprinting Characters (-v) **cat** is normally used for displaying text files only. Executables, when seen with **cat**, simply display junk. If you have nonprinting ASCII characters in your input, you can use **cat** with the -v option to display these characters.

Numbering Lines (-n) The -n option numbers lines. C compilers indicate the line number where errors are detected, and this numbering facility often helps a programmer in debugging programs. But then your **vi** editor can show line numbers too, and if your version of **cat** doesn't support -n, you can use the **pr** *(12.2)* command to do the same job.

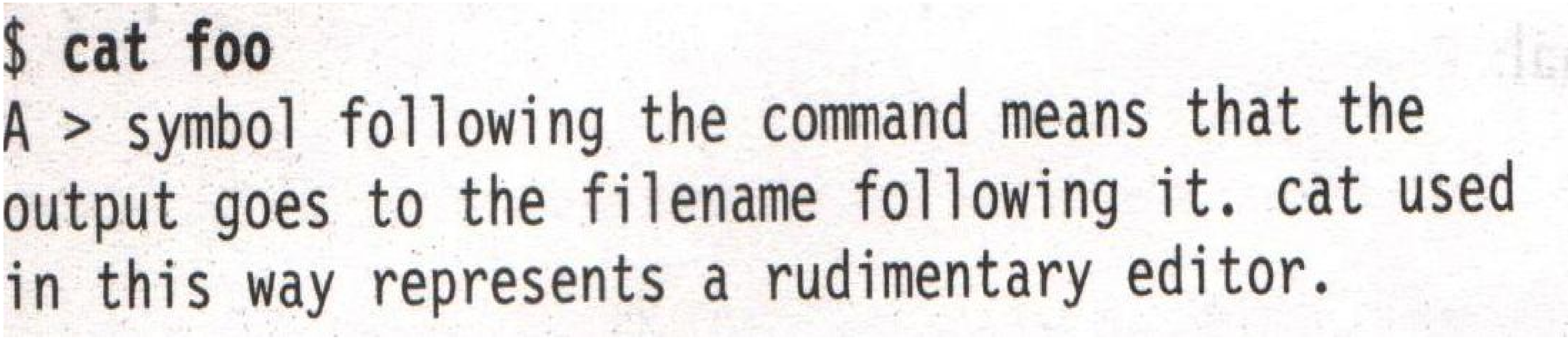
cat – DISPLAYING AND CREATING FILES

cat is also useful for creating files



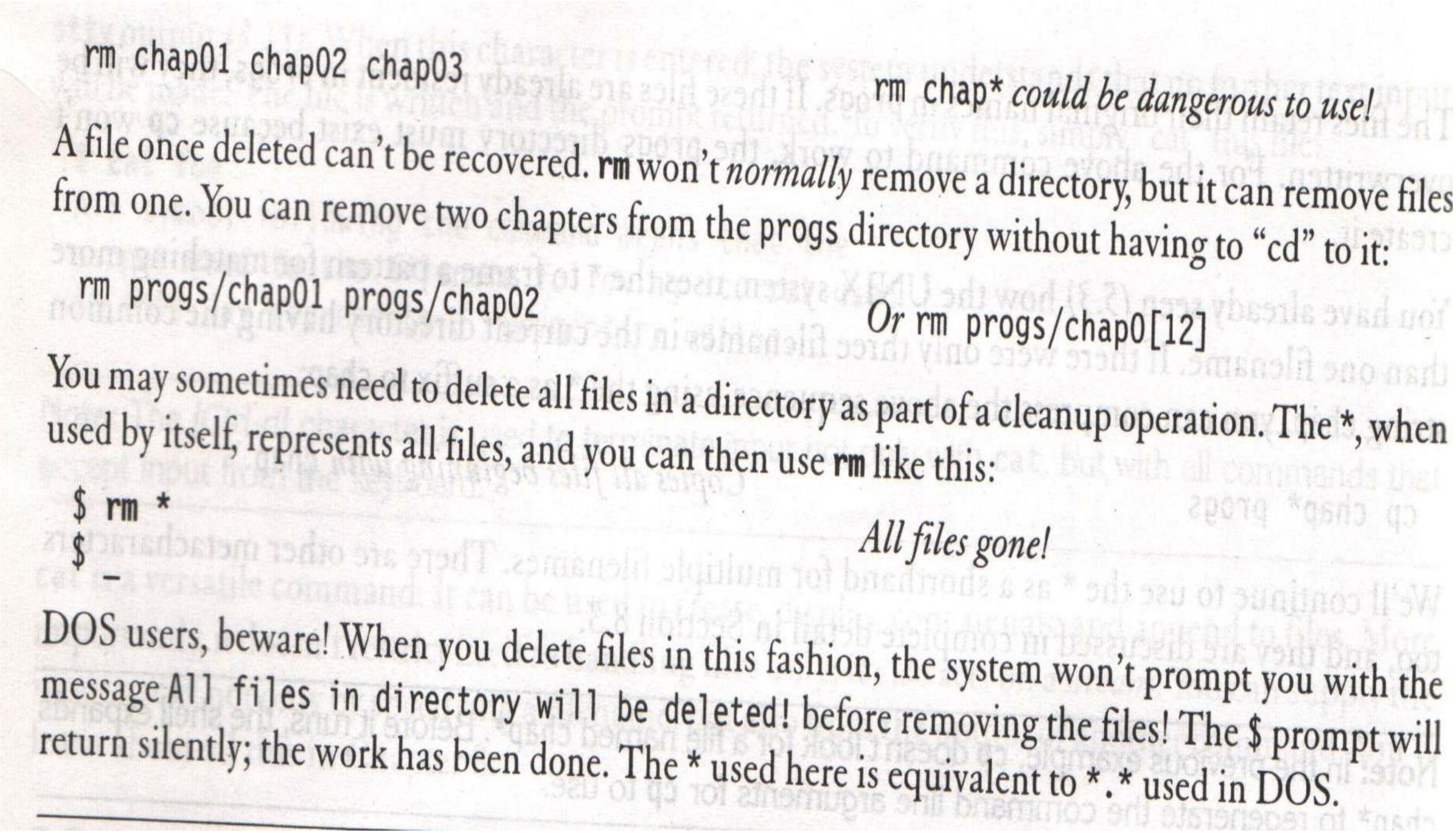
```
$ cat > foo
A > symbol following the command means that the
output goes to the filename following it. cat used
in this way represents a rudimentary editor.
[Ctrl-d]
$ _
```

Prompt returns



```
$ cat foo
A > symbol following the command means that the
output goes to the filename following it. cat used
in this way represents a rudimentary editor.
```

rm – DELETING FILES



```
rm chap01 chap02 chap03
```

`rm chap*` *could be dangerous to use!*

A file once deleted can't be recovered. **rm** won't *normally* remove a directory, but it can remove files from one. You can remove two chapters from the progs directory without having to "cd" to it:

```
rm progs/chap01 progs/chap02
```

Or `rm progs/chap0[12]`

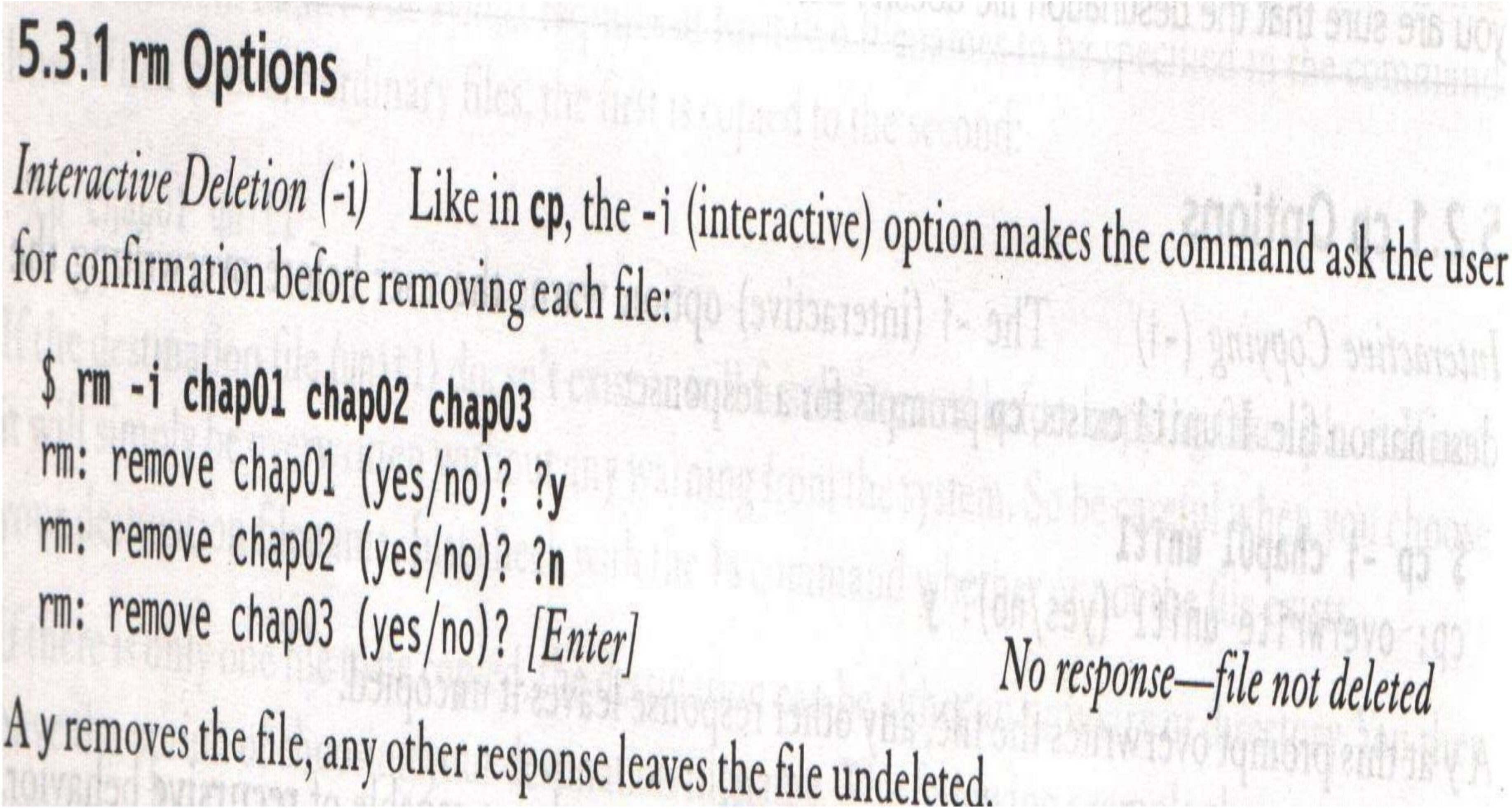
You may sometimes need to delete all files in a directory as part of a cleanup operation. The *, when used by itself, represents all files, and you can then use **rm** like this:

```
$ rm *
$ _
```

All files gone!

DOS users, beware! When you delete files in this fashion, the system won't prompt you with the message `All files in directory will be deleted!` before removing the files! The $ prompt will return silently; the work has been done. The * used here is equivalent to *.* used in DOS.

rm – DELETING FILES



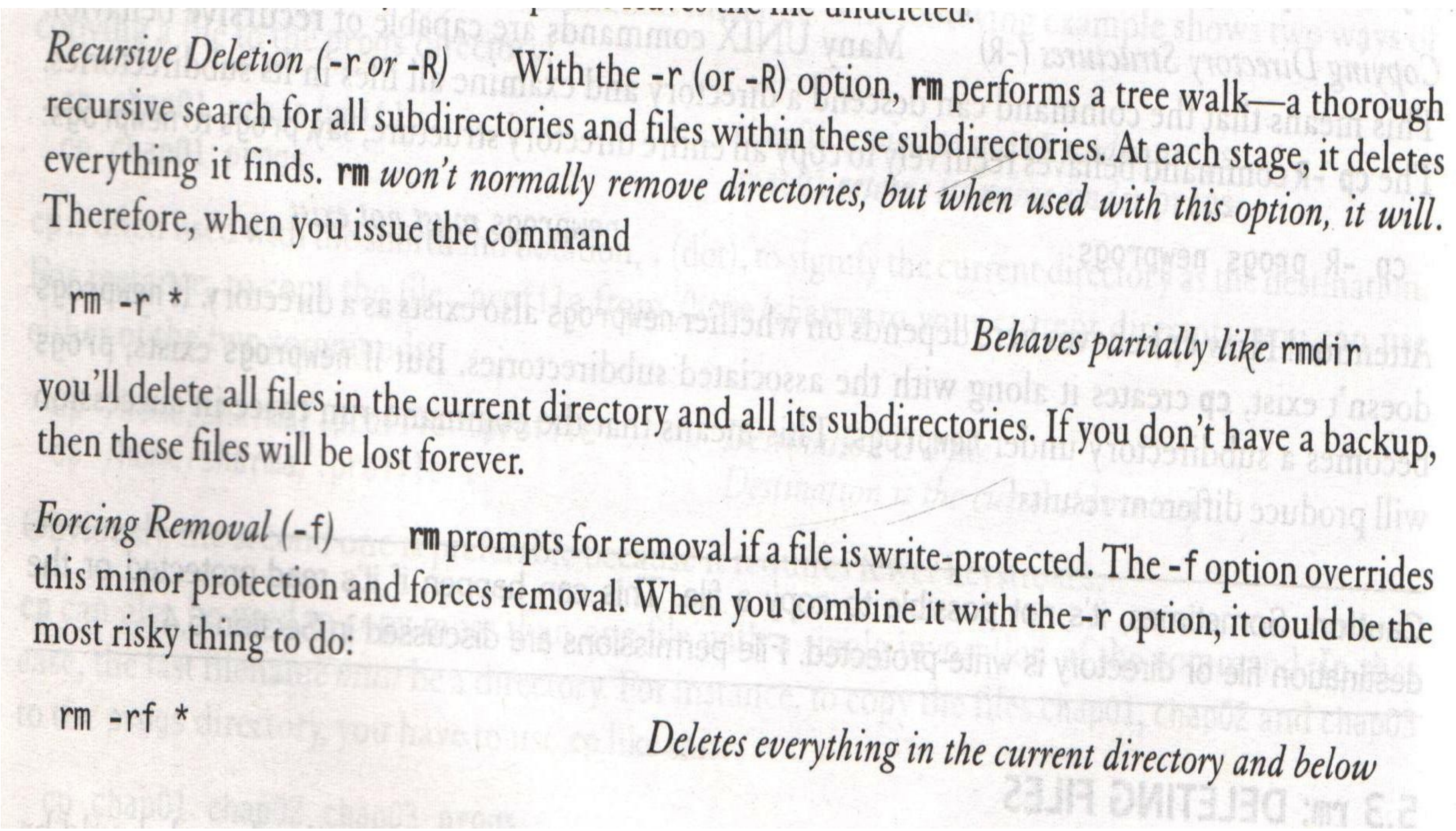
5.3.1 rm Options

Interactive Deletion (-i) Like in **cp**, the -i (interactive) option makes the command ask the user for confirmation before removing each file:

```
$ rm -i chap01 chap02 chap03
rm: remove chap01 (yes/no)? ?y
rm: remove chap02 (yes/no)? ?n
rm: remove chap03 (yes/no)? [Enter]
                                        No response—file not deleted
```

A y removes the file, any other response leaves the file undeleted.

rm – DELETING FILES



Recursive Deletion (`-r` *or* `-R`) With the `-r` (or `-R`) option, **rm** performs a tree walk—a thorough recursive search for all subdirectories and files within these subdirectories. At each stage, it deletes everything it finds. **rm** *won't normally remove directories, but when used with this option, it will.* Therefore, when you issue the command

```
rm -r *
```

Behaves partially like `rmdir`

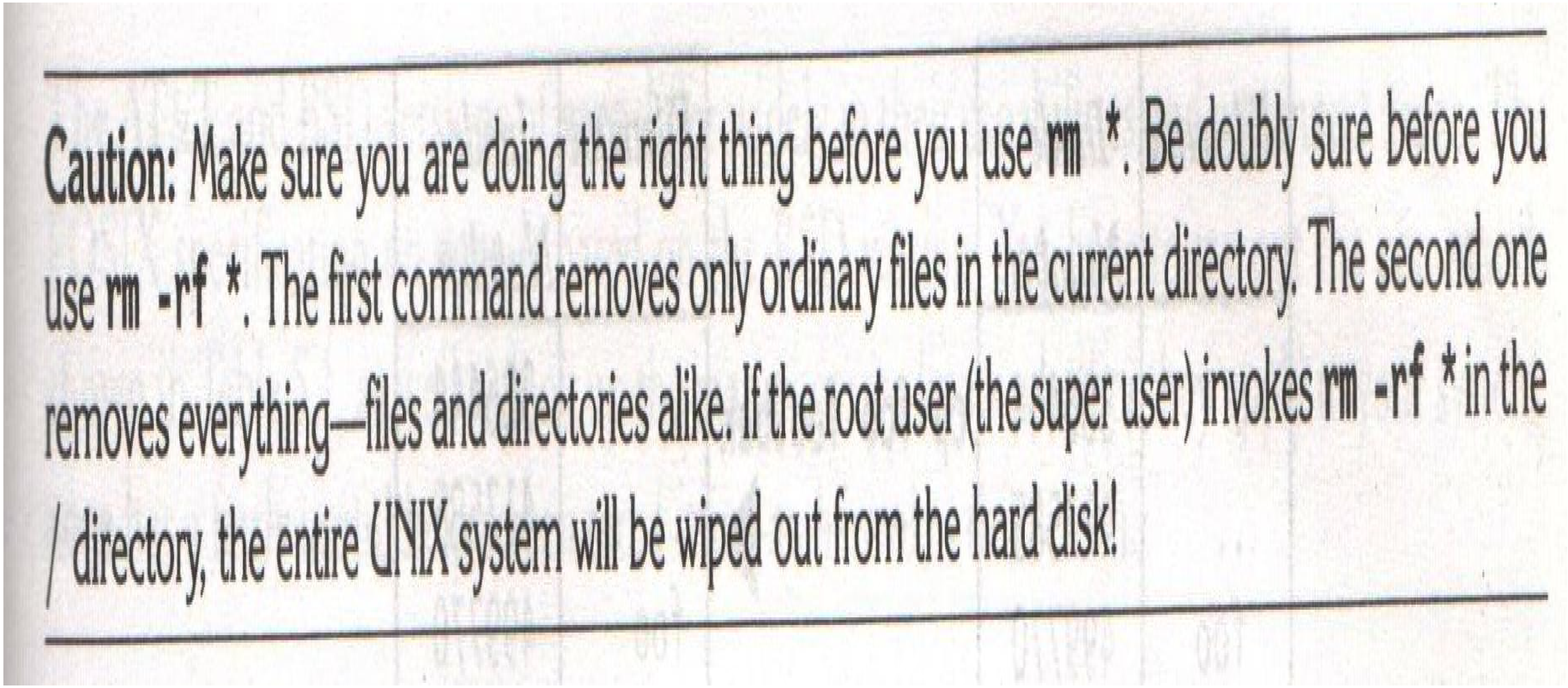
you'll delete all files in the current directory and all its subdirectories. If you don't have a backup, then these files will be lost forever.

Forcing Removal (`-f`) **rm** prompts for removal if a file is write-protected. The `-f` option overrides this minor protection and forces removal. When you combine it with the `-r` option, it could be the most risky thing to do:

```
rm -rf *
```

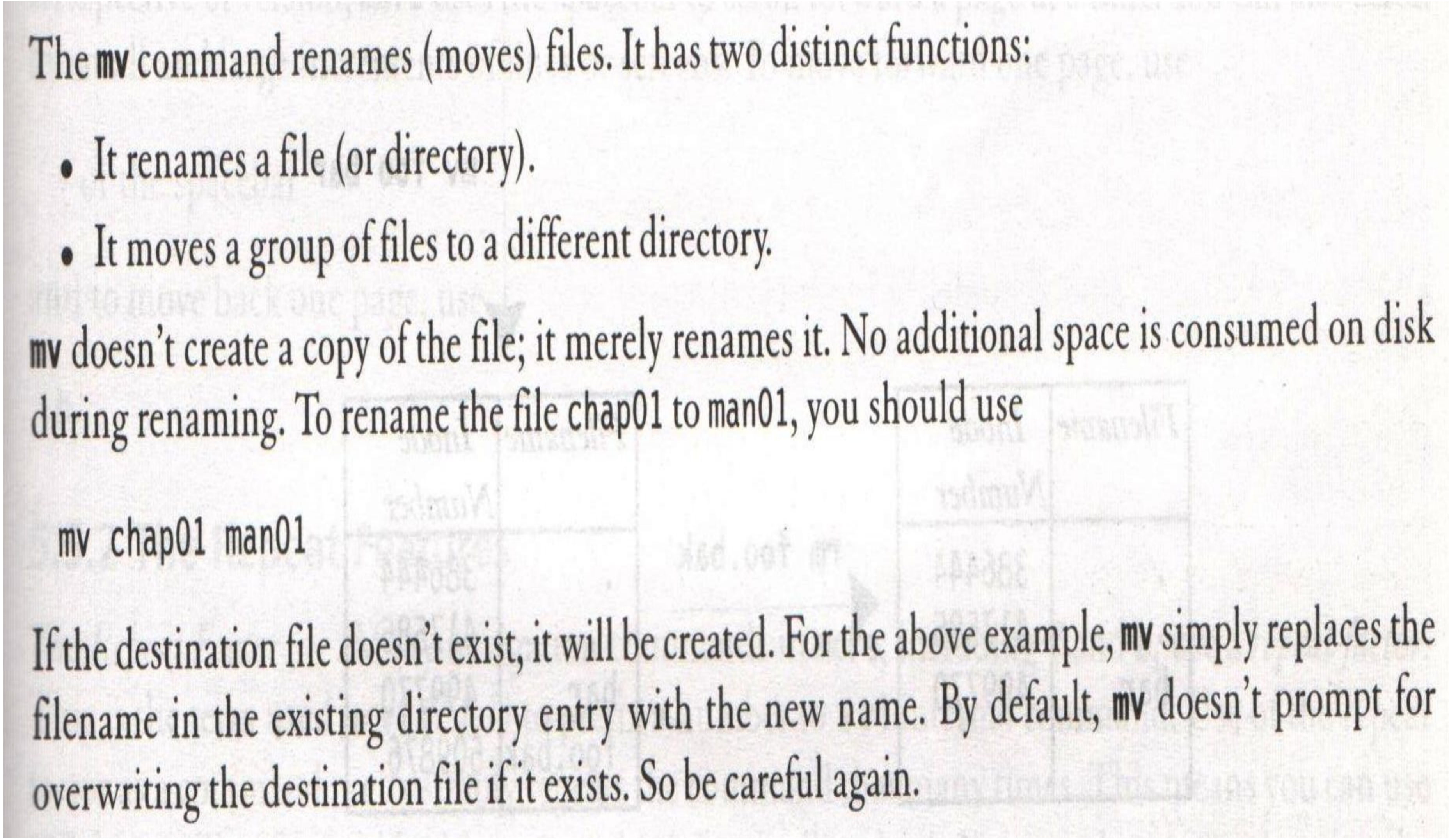
Deletes everything in the current directory and below

rm – DELETING FILES



Caution: Make sure you are doing the right thing before you use **`rm *`**. Be doubly sure before you use **`rm -rf *`**. The first command removes only ordinary files in the current directory. The second one removes everything—files and directories alike. If the root user (the super user) invokes **`rm -rf *`** in the / directory, the entire UNIX system will be wiped out from the hard disk!

mv – RENAMING FILES



The **mv** command renames (moves) files. It has two distinct functions:

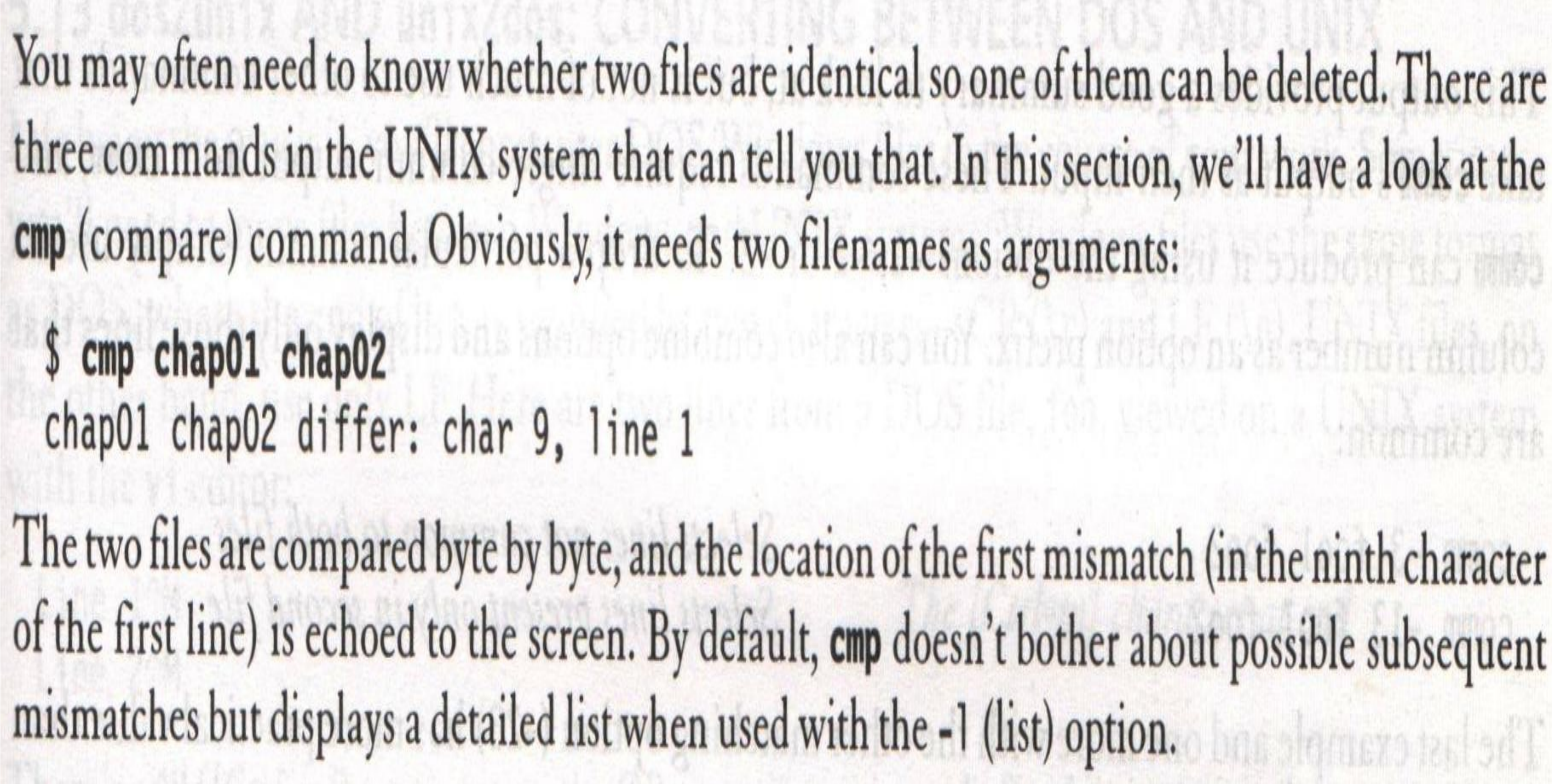
- It renames a file (or directory).
- It moves a group of files to a different directory.

mv doesn't create a copy of the file; it merely renames it. No additional space is consumed on disk during renaming. To rename the file `chap01` to `man01`, you should use

```
mv chap01 man01
```

If the destination file doesn't exist, it will be created. For the above example, **mv** simply replaces the filename in the existing directory entry with the new name. By default, **mv** doesn't prompt for overwriting the destination file if it exists. So be careful again.

cmp – COMPARING TWO FILES

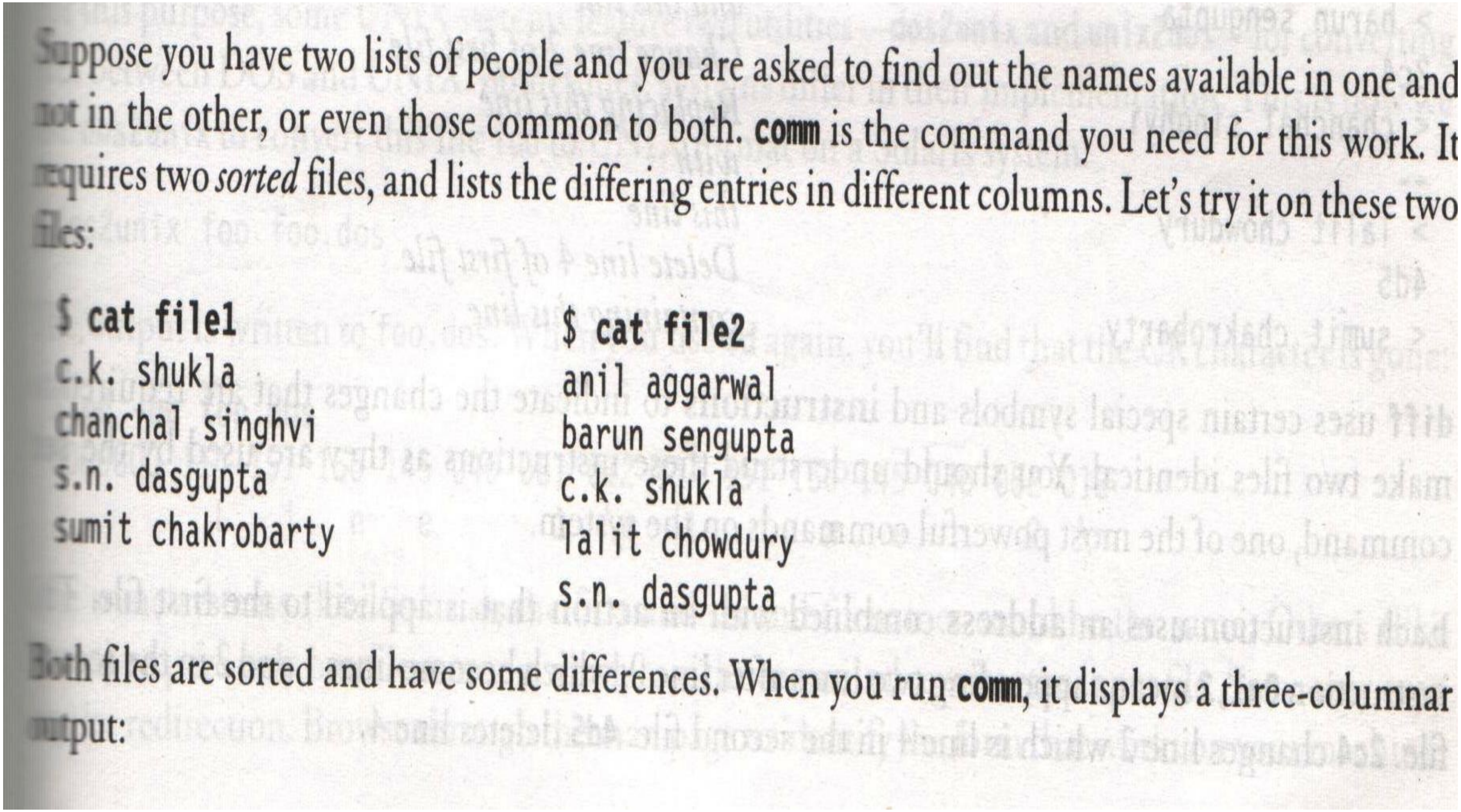


You may often need to know whether two files are identical so one of them can be deleted. There are three commands in the UNIX system that can tell you that. In this section, we'll have a look at the **cmp** (compare) command. Obviously, it needs two filenames as arguments:

```
$ cmp chap01 chap02
chap01 chap02 differ: char 9, line 1
```

The two files are compared byte by byte, and the location of the first mismatch (in the ninth character of the first line) is echoed to the screen. By default, **cmp** doesn't bother about possible subsequent mismatches but displays a detailed list when used with the **-l** (list) option.

comm – WHAT IS COMMON?

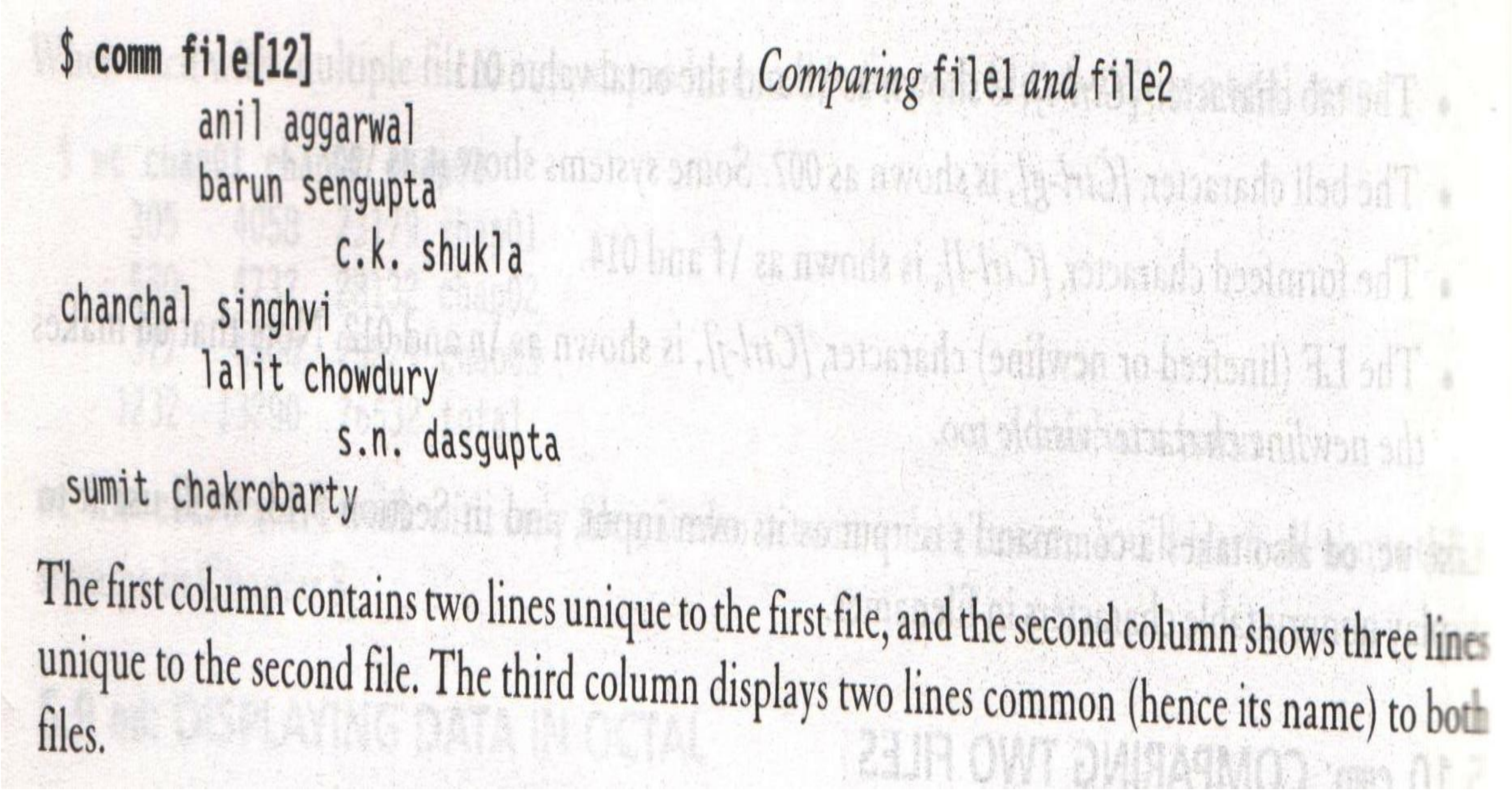


Suppose you have two lists of people and you are asked to find out the names available in one and not in the other, or even those common to both. **comm** is the command you need for this work. It requires two *sorted* files, and lists the differing entries in different columns. Let's try it on these two files:

```
$ cat file1                   $ cat file2
c.k. shukla                   anil aggarwal
chanchal singhvi              barun sengupta
s.n. dasgupta                 c.k. shukla
sumit chakrobarty             lalit chowdury
                              s.n. dasgupta
```

Both files are sorted and have some differences. When you run **comm**, it displays a three-columnar output:

comm – WHAT IS COMMON?

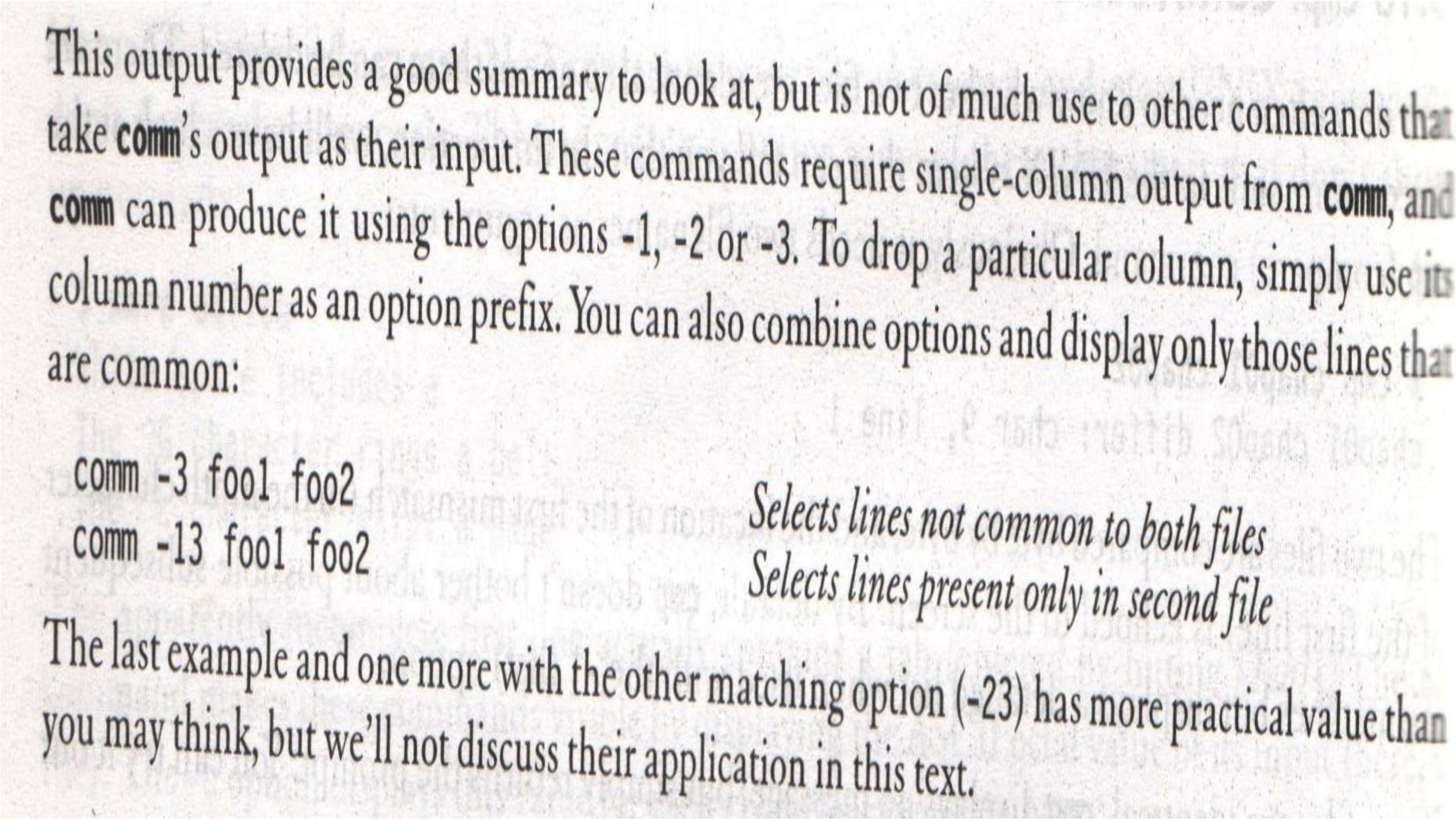


Comparing file1 *and* file2

```
$ comm file[12]
        anil aggarwal
        barun sengupta
                c.k. shukla
chanchal singhvi
        lalit chowdury
                s.n. dasgupta
sumit chakrobarty
```

The first column contains two lines unique to the first file, and the second column shows three lines unique to the second file. The third column displays two lines common (hence its name) to both files.

comm – WHAT IS COMMON?

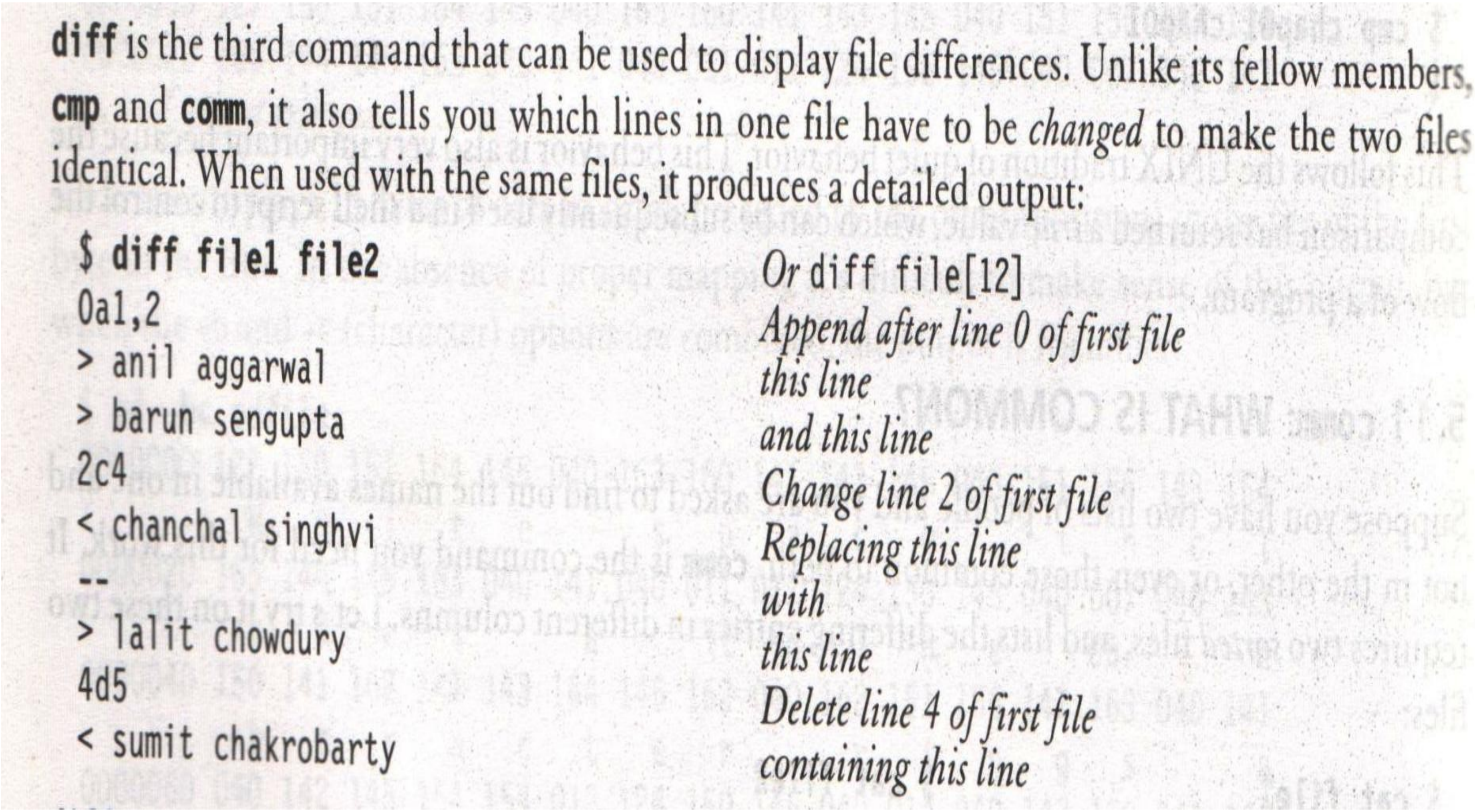


This output provides a good summary to look at, but is not of much use to other commands that take **comm**'s output as their input. These commands require single-column output from **comm**, and **comm** can produce it using the options -1, -2 or -3. To drop a particular column, simply use its column number as an option prefix. You can also combine options and display only those lines that are common:

`comm -3 foo1 foo2`	*Selects lines not common to both files*
`comm -13 foo1 foo2`	*Selects lines present only in second file*

The last example and one more with the other matching option (-23) has more practical value than you may think, but we'll not discuss their application in this text.

diff – CONVERTING ONE FILE TO OTHER



diff is the third command that can be used to display file differences. Unlike its fellow members, **cmp** and **comm**, it also tells you which lines in one file have to be *changed* to make the two files identical. When used with the same files, it produces a detailed output:

```
$ diff file1 file2                  Or diff file[12]
0a1,2                               Append after line 0 of first file
> anil aggarwal                     this line
> barun sengupta                    and this line
2c4                                 Change line 2 of first file
< chanchal singhvi                  Replacing this line
--                                  with
> lalit chowdury                    this line
4d5                                 Delete line 4 of first file
< sumit chakrobarty                 containing this line
```

umask: DEFAULT FILE AND DIRECTORY PERMISSIONS

- Usually we have,

  rw-rw-rw- (octal 666) for regular files

  rwxrwxrwx (octal 777) for directories
- The default is transformed by subtracting the user mask from it to remove one or more permissions
- We can evaluate the current value of the mask as,

```
$ umask
  022
```

The umask (UNIX shorthand for "user file-creation mode mask") is **a four-digit octal number that UNIX uses to determine the file permission for newly created files**. Every process has its own umask, inherited from its parent process.

FILE OWNERSHIP

- When you create a file, you become its owner (third column)
- Group owner of the file (fourth column)
- Several users may belong to a single group, but the privileges of the group are set by the owner of the file and not by the group members

FILE OWNERSHIP

- UNIX follows a three-tiered file protection system that determines a file’s access rights
- Filetype owner (rwx) groupowner (rwx) others (rwx)
- Example:

-rwxr-xr-- 1 kumar metal 20500 may 10 19:21 chap02

r w x r - x

r - -

owner/user group owner others

CHANGING FILE PERMISSIONS

- A file or a directory is created with a default set of permissions, which can be determined by umask

- Let us assume that the file permission for the created file is -rw-r--r—

- Using **chmod** command, we can change the file permissions and allow the owner to execute his file

RELATIVE AND ABSOLUTE PERMISSIONS

- In a relative manner, specify the changes to the current permissions

- In an absolute manner, specify the final permissions

RELATIVE PERMISSIONS

- chmod only changes the permissions specified in the command line and **leaves the other permissions unchanged**
- Syntax

  chmod category operation permission filename(s)

RELATIVE PERMISSIONS

- chmod takes an expression as its argument which contains:

1. user category (user, group, others)
2. operation to be performed (assign or remove a permission)
3. type of permission (read, write, execute)

RELATIVE PERMISSIONS

Category	**operation**	**permission**
u - user	+ assign	r - read
g - group	- remove	w - write
o - others	= absolute	x - execute
a - all (ugo)		

RELATIVE PERMISSIONS

- Examples

```
-rw-r--r-- 1      kumar  metal  1906   sep      23:38
   xstart
```

```
chmod u+x xstart
```

```
-rwxr--r-- 1      kumar  metal  1906   sep      23:38
   xstart
```

The command assigns (+) execute (x) permission to the user (u), other permissions remain unchanged

RELATIVE PERMISSIONS

- chmod ugo+x xstart
- chmod a+x xstart
- chmod +x xstart

```
-rwxr-xr-x 1    kumar  metal  1906   sep      23:38   xstart
```

chmod accepts multiple file names in command line

- chmod u+x note note1 note3

RELATIVE PERMISSIONS

Let initially

```
-rwxr-xr-x 1   kumar   metal   1906   sep 23:38    xstart
```

```
chmod go-r xstart
```

Then, it becomes

```
-rwx--x--x 1   kumar   metal   1906   sep 23:38    xstart
```

ABSOLUTE PERMISSIONS

- Need not to know the current file permissions
- Set all nine permissions explicitly
- A string of three octal digits is used as an expression
- The permission can be represented by one octal digit for each category
- For each category, we add octal digits

ABSOLUTE PERMISSIONS

Octal	Permissions	Significance
0	- - -	no permissions
1	- - x	execute only
2	- w -	write only
3	- w x	write and execute
4	r - -	read only
5	r - x	

ABSOLUTE PERMISSIONS

- Using relative permission

```
chmod a+rw xstart
```

- Using absolute permission

```
chmod 666 xstart
chmod 644 xstart
chmod 761 xstart
```

will assign all permissions to the owner, read and write permissions for the group and only execute permission to the others

ABSOLUTE PERMISSIONS

- 777 signifies all permissions for all category, but still we can prevent a file from being deleted

- 000 signifies absence of all permissions for all category, but still we can delete a file

- It is the directory permissions that determines whether a file can be deleted or not

PERMISSIONS

- Only owner can change the file permissions
- User can not change other user’s file’s permissions
- But the system administrator can do anything

THE SECURITY IMPLICATIONS

- Let the default permission for the file xstart is
  -rw-r--r--
- chmod u-rw, go-r xstart   or
- chmod 000 xstart

- This is simply useless but still the user can delete this file

chmod RECURSIVELY

```
chmod -R a+x shell_scripts
```

This makes all the files and subdirectories found in the shell_scripts directory, executable by all users

DIRECTORY PERMISSIONS

- It is possible that a file cannot be accessed even though it has read permission, and can be removed even when it is write protected
- The default permissions of a directory are

  rwxr-xr-x (755)
- A directory must never be writable by group and others

DIRECTORY PERMISSIONS

This becomes 644 (666-022) for ordinary files and 755 (777-022) for directories

umask 000

Indicates, we are not subtracting anything and the default permissions will remain unchanged

Note that, changing system wide default permission seetings is possible using chmod but not by umask

CHANGING FILE OWNERSHIP

- On BSD and AT&T systems
- Kumar – owner (user)
- Metal – group owner
- If sharma copies a file of kumar, then sharma will become its owner and he can manipulate the attributes
- chown and chgrp
- On BSD, only system administrator can use chown
- On other systems, only the owner can change both

chown Command

- Changing ownership requires superuser permission, so use **su** command

  ```
  ls -l note
  -rwxr----x 1 kumar  metal  347  may  10  20:30  note
  ```

- chown sharma note; ls -l note

  ```
  -rwxr----x 1 sharma  metal  347  may  10  20:30  note
  ```

chgrp Command

- This command changes the file’s group owner
- No superuser permission is required

```
         ls –l dept.lst

-rw-r--r--  1  kumar  metal  139  jun  8  16:43  dept.lst
         chgrp  dba  dept.lst;
         ls –l dept.lst
-rw-r--r--  1  kumar  dba  139  jun  8  16:43  dept.lst
```

sort : ordering a file

Sorting is the ordering of data in ascending or descending sequence. The sort command orders a file and by default, the entire line is sorted

This default sorting sequence can be altered by using certain options. We can also sort one or more keys (fileds) or use a different ordering rule.

sort options

The important sort options are:

-k n	**sorts on nth field**
-k m,n	**starts sort on mth field and ends sort on nth field**
-k m.n	**starts sort on nth column of mth field**
-u	**removes repeated lines**
-n	**sorts numerically**
-r	**reverses sort order**
-m list	**merges sorted files in list**
-c	**checks if file is sorted**
-o flname	**places output in file flname**

sort : ordering a file

```
sort –t“|” –k 2 shortlist
```

sorts the second field (name)

```
sort –t”|” –r –k 2 shortlist
```

or

```
sort –t”|” –k 2r shortlist
```

sort order can be revered with this –r option.

```
sort –t”|” –k 3,3 –k 2,2 shortlist
```

sorting on secondary key is also possible as shown above.

```
sort –t”|” –k 5.7,5.8 shortlist
```

we can also specify a character position with in a field to be the beginning of sort as shown above (sorting on columns).

head - display the start of a file

head displays the head, or start, of the file.

Syntax

head [options] file

Common Options

-n number number of lines to display, counting from the top of the file

-number same as above

Examples

By default **head** displays the first 10 lines. You can display more with the "**-n number**", or "**-number**" options, e.g., to display the first 40 lines:

```
head -40 filename or head -n 40 filename
```

tail - display the end of a file

tail displays the tail, or end, of the file.

Syntax

tail [options] file

Common Options

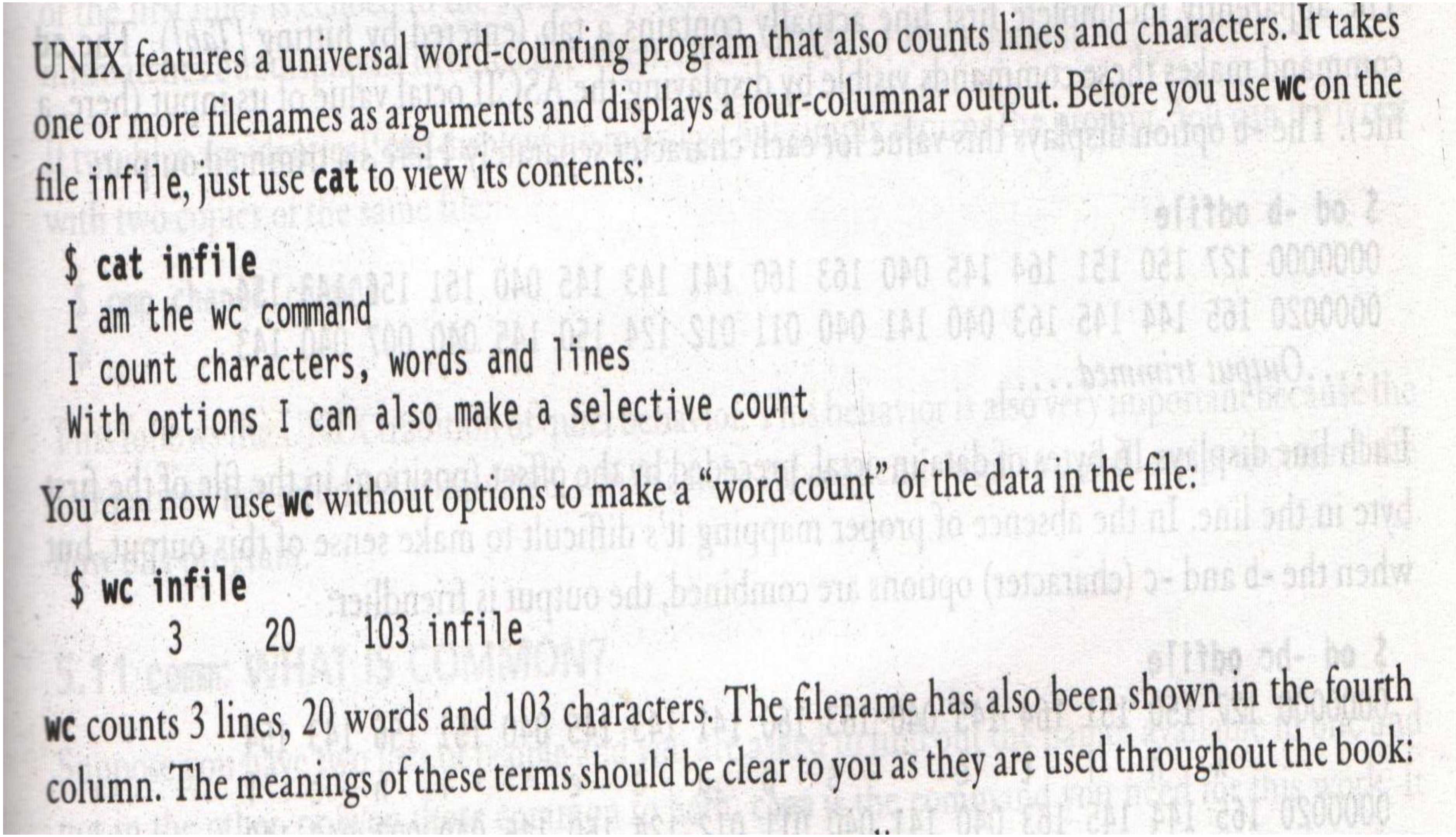
-number number of lines to display, counting from the bottom of the file

Examples

The default is to display the last 10 lines, but you can specify different line or byte numbers, or a different starting point within the file. To display the last 30 lines of a file use the **-number** style:

```
tail -30 filename
```

wc – COUNTING LINES, WORDS AND CHARACTERS



UNIX features a universal word-counting program that also counts lines and characters. It takes one or more filenames as arguments and displays a four-columnar output. Before you use **wc** on the file `infile`, just use **cat** to view its contents:

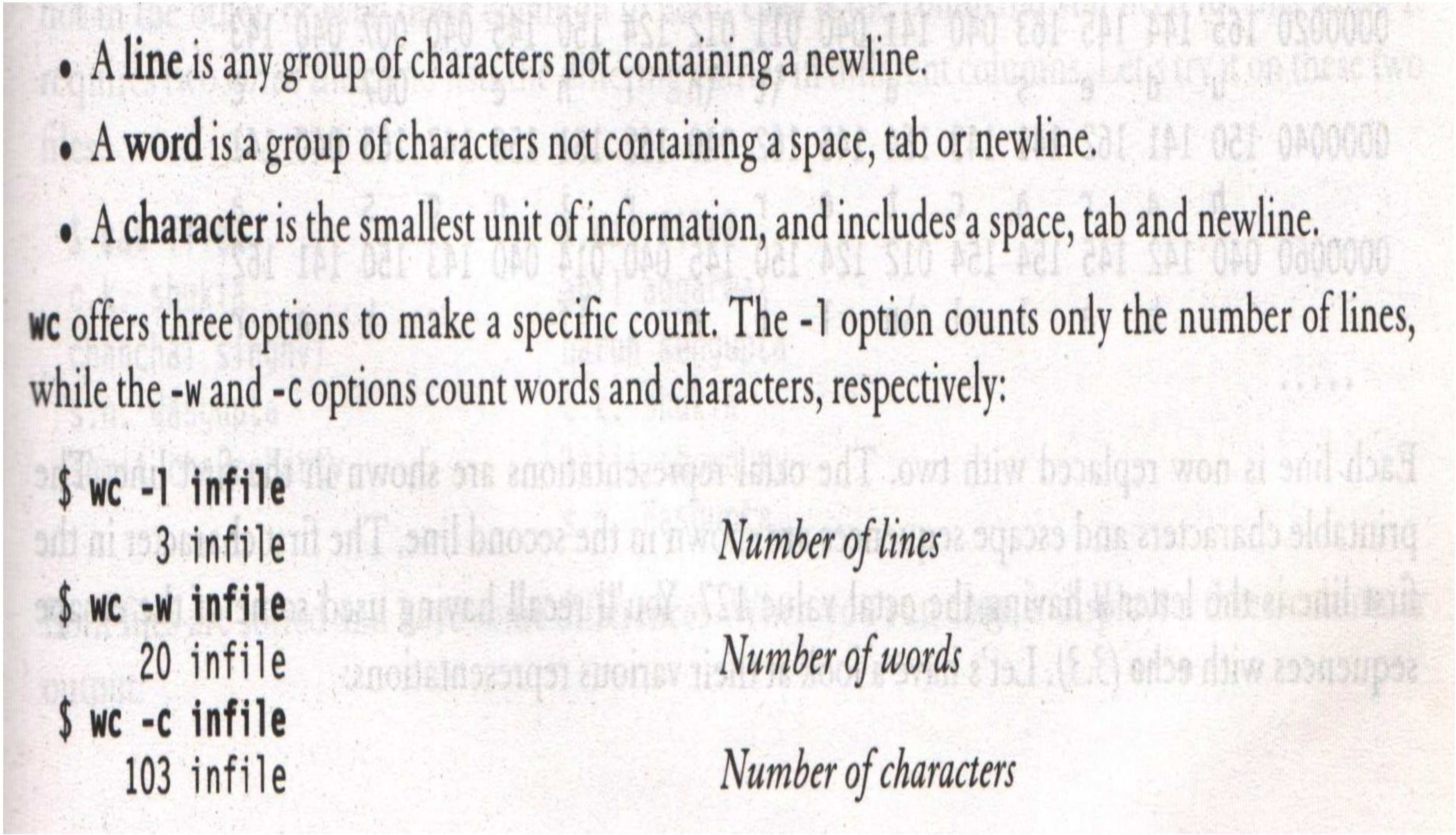
```
$ cat infile
I am the wc command
I count characters, words and lines
With options I can also make a selective count
```

You can now use **wc** without options to make a "word count" of the data in the file:

```
$ wc infile
       3      20     103 infile
```

wc counts 3 lines, 20 words and 103 characters. The filename has also been shown in the fourth column. The meanings of these terms should be clear to you as they are used throughout the book:

wc – COUNTING LINES, WORDS AND CHARACTERS

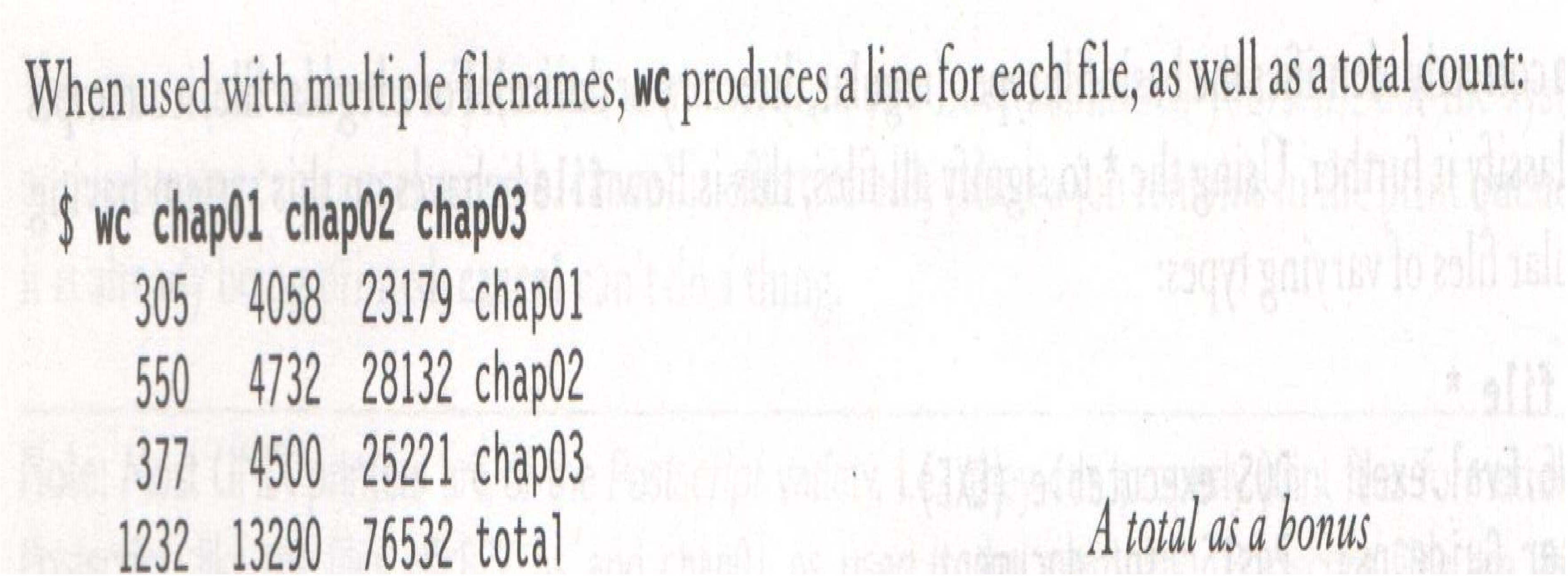


- A **line** is any group of characters not containing a newline.
- A **word** is a group of characters not containing a space, tab or newline.
- A **character** is the smallest unit of information, and includes a space, tab and newline.

`wc` offers three options to make a specific count. The `-l` option counts only the number of lines, while the `-w` and `-c` options count words and characters, respectively:

```
$ wc -l infile
      3 infile                    Number of lines
$ wc -w infile
     20 infile                    Number of words
$ wc -c infile
    103 infile                    Number of characters
```

wc – COUNTING LINES, WORDS AND CHARACTERS



When used with multiple filenames, **wc** produces a line for each file, as well as a total count:

```
$ wc chap01 chap02 chap03
     305    4058   23179 chap01
     550    4732   28132 chap02
     377    4500   25221 chap03
    1232   13290   76532 total
```

A total as a bonus

Tee Command

Tee command is used to store and view (both at the same time) the output of any other command.

Tee command writes to the STDOUT, and to a file at a time as shown in the examples below

The following command writes the output only to the file and not to the screen.

```
$ ls > file
```

The following command (with the help of tee command) writes the output both to the screen (stdout) and to the file.

```
$ ls | tee file
```

top - display top CPU processes

SYNOPSIS

top [**-**] [**d** *delay*] [**p** *pid*] [**q**] [**c**] [**C**] [**S**] [**s**] [**i**] [**n** *iter*] [**b**] **DESCRIPTION**

top provides an ongoing look at processor activity in real time.

It displays a listing of the most CPU-intensive tasks on the system, and can provide an interactive interface for manipulating processes.

It can sort the tasks by CPU usage, memory usage and runtime. can be better configured than the standard top from the procps suite.

Most features can either be selected by an interactive command or by specifying the feature in the personal or system-wide configuration file. See below for more information.

grep Command

grep to search a file(s) for a pattern and display.

grep options to display, count, line numbers or filenames

Regular expressions

- Basic regular expressions (BRE)
- Extended regular expressions (ERE)
- **sed** to edit / manipulate an input stream
- substitution features
- repeated and remembered patterns

grep Command

grep : searching for a pattern

grep options

Basic regular expressions (BRE)

- An introduction
- The character class
- The *
- The dot
- Specifying pattern locations Metacharacters

grep Command

- It scans the file / input for a pattern and displays lines containing the pattern, the line numbers or filenames where the pattern occurs

- It's a command from a special family in UNIX for handling search requirements

grep *options pattern filename(s)*

emp.lst (Data used for grep)

- 2233 | a.k.shukla | g.m | sales | 12/12/52 | 6000
- 9876 | jai sharma | director | production | 12/03/50 | 7000
- 5678 | sumit chakrobarty | d.g.m. | marketing | 19/04/43 | 6000
- 2365 | barun sengupta | director | personnel | 11/05/47 | 7800
- 5423 | n.k.gupta | chairman | admin | 30/08/56 | 5400
- 1006 | chanchal singhvi | director | sales | 03/09/38 | 6700
- 6213 | karuna ganguly | g.m. | accounts | 05/06/62 | 6300
- 1265 | s.n. dasgupta | manager | sales | 12/09/63 | 5600
- 4290 | jayant choudhury | executive | production | 07/09/50 | 6000
- 2476 | anil aggarwal | manager | sales | 01/05/59 | 5000
- 6521 | lalit chowdury | directir | marketing | 26/09/45 | 8200
- 3212 | shyam saksena | d.g.m. | accounts | 12/12/55 | 6000
- 3564 | sudhir agarwal | executive | personnel | 06/07/47 | 7500
- 2345 | j. b. sexena | g.m. | marketing | 12/03/45 | 8000
- 0110 | v.k.agrawal | g.m.| marketing | 31/12/40 | 9000

grep Command

```
grep “sales” emp.lst
```

- Patterns with and without quotes is possible
- Its generally safe to quote the pattern
- Quote is mandatory when pattren involves more than one word
- It returns the prompt in case the pattren can’t be located

```
grep president emp.lst
```

grep Command

- When grep is used with multiple filenames, it displays the filenames along with the output

```
grep “director” emp1.lst emp2.lst
```

Where it shows filename followed by the contents

grep frequently used options

-i	ignores case for matching
-v	doesn’t display lines matching expression
-n	displays line numbers along with lines
-c	displays count of number of occurrences
-l	displays list of filenames only

grep Command

-e exp	specifies expression with this option
-x	matches pattern with entire line
-f file	takes pattrens from file, one per line
-E	treats pattren as an extended RE
-F	matches multiple fixed strings

grep Command

1. grep -i ‘agarwal’ emp.lst
2. grep -v ‘director’ emp.lst > otherlist

   wc -l otherlist will display 11 otherlist

1. grep –n ‘marketing’ emp.lst
2. grep –c ‘director’ emp.lst
3. grep –c ‘director’ emp*.lst

   will print filenames prefixed to the line count

grep Command

1.
   ```
   grep -i ‘manager’ *.lst
   ```

   will display filenames *only*

2.
   ```
   grep -e 'Agarwal' -e 'aggarwal' -e 'agrawal' emp.lst
   ```

   will print matching multiple patterns

BASIC REGULAR EXPRESSIONS

- It is tedious to specify each pattern separately with the -e option
- grep uses an expression of a different type to match a group of similar patterns
- if an expression uses meta characters, it is termed a regular expression
- Some of the characters used by regular expression are also meaningful to the shell

BRE character subset

*****	Zero or more occurrences
g*	nothing or g, gg, ggg, etc.
.	A single character
.*	nothing or any number of characters
[pqr]	a single character p, q or r
[c1-c2]	a single character within the ASCII range represented by c1 and c2

The character class

- grep supports basic regular expressions (BRE) by default and extended regular expressions (ERE) with the –E option

- A regular expression allows a group of characters enclosed within a pair of [ ], in which the match is performed for a single character in the group

grep Command

```
grep “[aA]g[ar][ar]wal” emp.lst
```

- A single pattern has matched two similar strings
- The pattern [a-zA-Z0-9] matches a single alphanumeric character. When we use range, make sure that the character on the left of the hyphen has a lower ASCII value than the one on the right

Negating a class (^) (caret)

THE *

* Zero or more occurrences of the previous character

g* nothing or g, gg, ggg, etc.

`grep “[aA]gg*[ar][ar]wal” emp`

Notice that we don’t require to use –e option three times to get the same output!!!!!

THE DOT

A dot matches a single character

.* signifies any number of characters or none

```
grep “j.*saxena” emp.lst
```

^ and $

Most of the regular expression characters are used for matching patterns, but there are two that can match a pattern at the beginning or end of a line

^	for matching at the beginning of a line
$	for matching at the end of a line

grep Command

```
grep “^2” emp.lst
```

Selects lines where emp_id starting with 2

```
grep “7…$” emp.lst
```

Selects lines where emp_salary ranges between 7000 to 7999

```
grep “^[^2]” emp.lst
```

Selects lines where emp_id doesn’t start with 2

When meta characters lose their meaning

- It is possible that some of these special characters actually exist as part of the text
- Sometimes, we need to escape these characters

Eg: when looking for a pattern g*, we have to use \

To look for [, we use \[

To look for .*, we use \.*

EXTENDED RE (ERE)

- If current version of grep doesn’t support ERE, then use egrep but without the –E option
- -E option treats pattern as an ERE

+ matches one or more occurrences of the previous character

? Matches zero or one occurrence of the previous

character

grep ERE

b+ matches b, bb, bbb, etc.

b? matches either a single instance of b or nothing

These characters restrict the scope of match as compared to the *

```
grep –E “[aA]gg?arwal” emp.lst
```

The ERE set

ch+	matches one or more occurrences of character ch
ch?	Matches zero or one occurrence of character ch
exp1|exp2	matches exp1 or exp2
(x1|x2)x3	matches x1x3 or x2x3

Matching multiple patterns

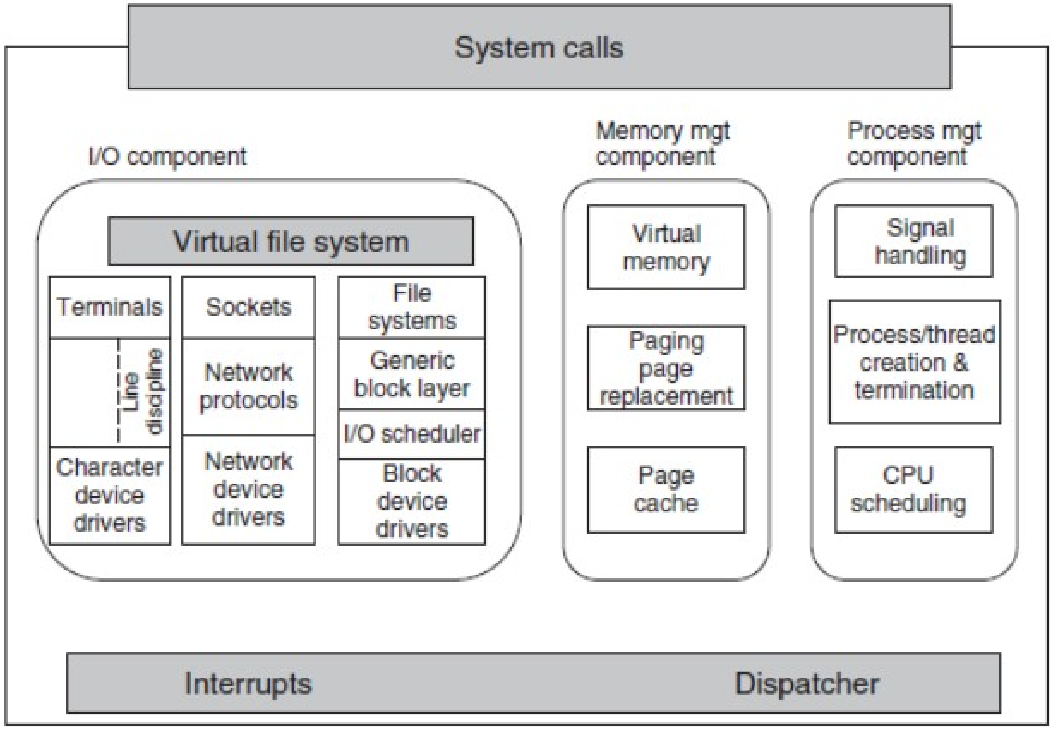
```
grep -E 'sengupta|dasgupta' emp.lst
```

We can locate both without using –e option twice, or

```
grep -E '(sen|das)gupta' emp.lst
```

Linux Kernel

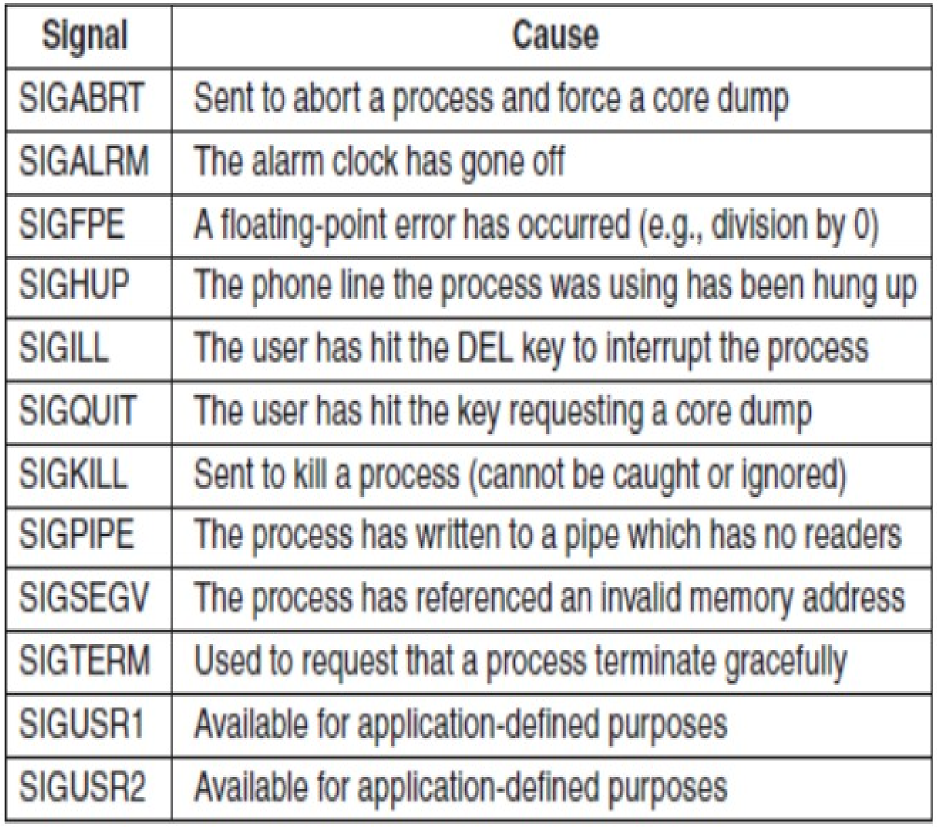
- It is a Unix-like computer operating system kernel.
- The Linux kernel API, the Application Programming Interface (API) through which user programs interact with the kernel, is meant to be very stable and to not break user space programs.



System calls
I/O component
Memory mgt component
Process mgt component
Virtual file system
Terminals
Sockets
File systems
Line discipline
Network protocols
Generic block layer
I/O scheduler
Character device drivers
Network device drivers
Block device drivers
Virtual memory
Paging page replacement
Page cache
Signal handling
Process/thread creation & termination
CPU scheduling
Interrupts
Dispatcher

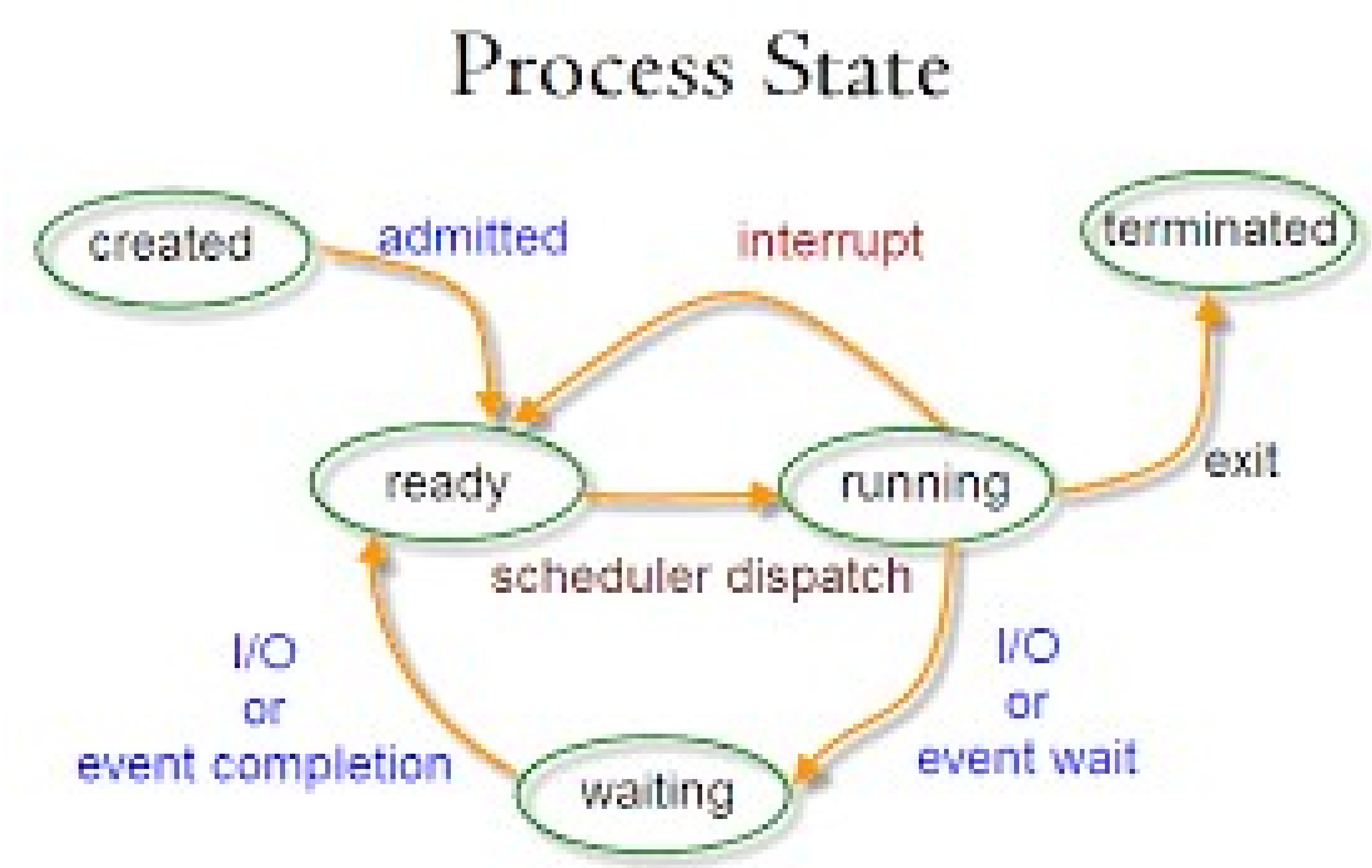
Process in Linux and PID, UID, GID in Linux

- The fork system call creates an exact copy of the original process.
- The forking process is called the parent process. The new process is called the child process.
- Process Identifier is when each process has a unique identifier associated with it known as process id.
- User and Group Identifiers (UID and GID) are the identifiers associated with a processes of the user and group.



Signal	Cause
SIGABRT	Sent to abort a process and force a core dump
SIGALRM	The alarm clock has gone off
SIGFPE	A floating-point error has occurred (e.g., division by 0)
SIGHUP	The phone line the process was using has been hung up
SIGILL	The user has hit the DEL key to interrupt the process
SIGQUIT	The user has hit the key requesting a core dump
SIGKILL	Sent to kill a process (cannot be caught or ignored)
SIGPIPE	The process has written to a pipe which has no readers
SIGSEGV	The process has referenced an invalid memory address
SIGTERM	Used to request that a process terminate gracefully
SIGUSR1	Available for application-defined purposes
SIGUSR2	Available for application-defined purposes

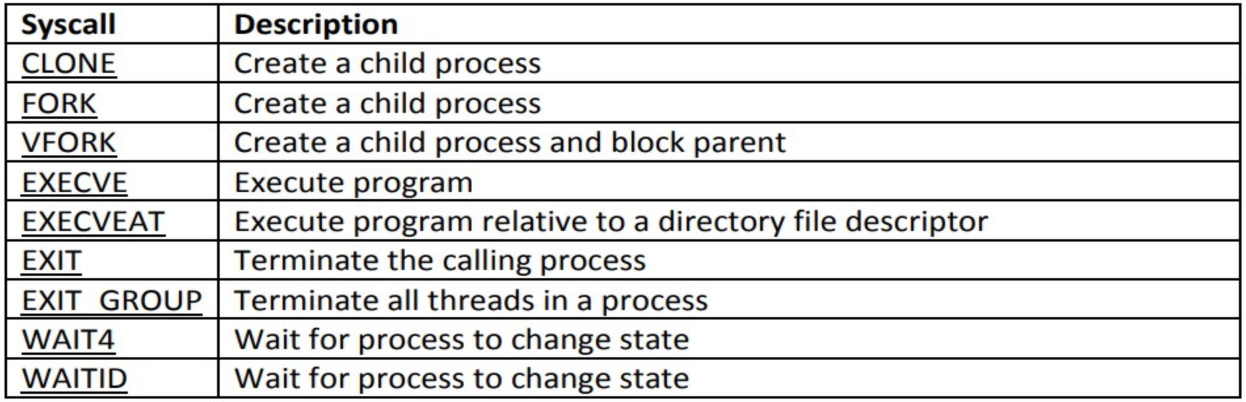
Linux Processes


Process State
created
admitted
interrupt
terminated
ready
running
exit
scheduler dispatch
I/O
or
event completion
I/O
or
event wait
waiting

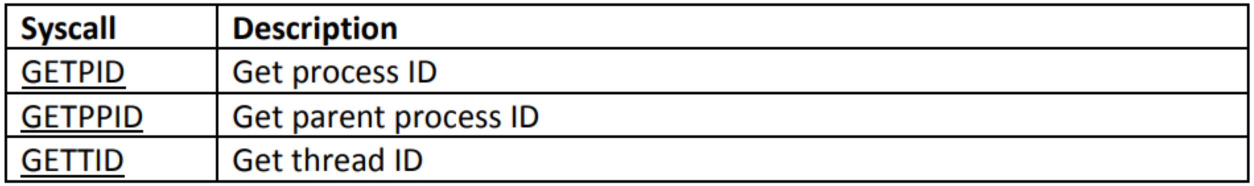
Linux Processes

- State
- Running
- Waiting
- Stopped
- Zombie – dead processes (not removed)
- Scheduling Information
- Identifiers
- Inter-Process Communication
- Links
- Times and Timers
- Virtual memory
- Processor Specific Context
- File system

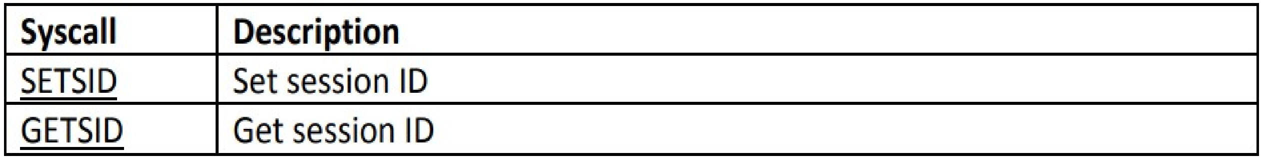
Process Management System Calls in Linux



Syscall	Description
CLONE	Create a child process
FORK	Create a child process
VFORK	Create a child process and block parent
EXECVE	Execute program
EXECVEAT	Execute program relative to a directory file descriptor
EXIT	Terminate the calling process
EXIT_GROUP	Terminate all threads in a process
WAIT4	Wait for process to change state
WAITID	Wait for process to change state

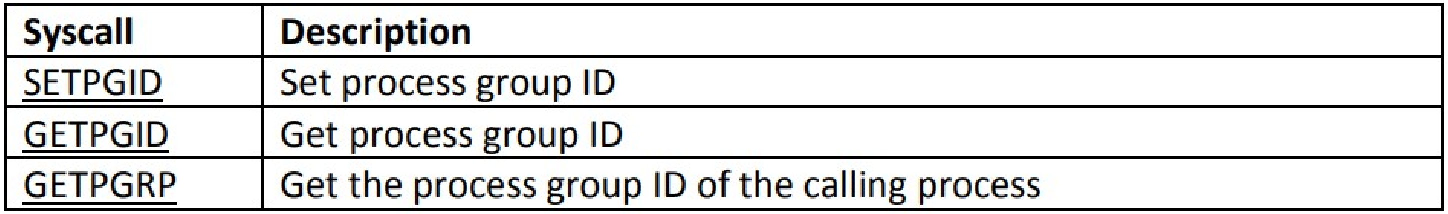


Syscall	Description
GETPID	Get process ID
GETPPID	Get parent process ID
GETTID	Get thread ID

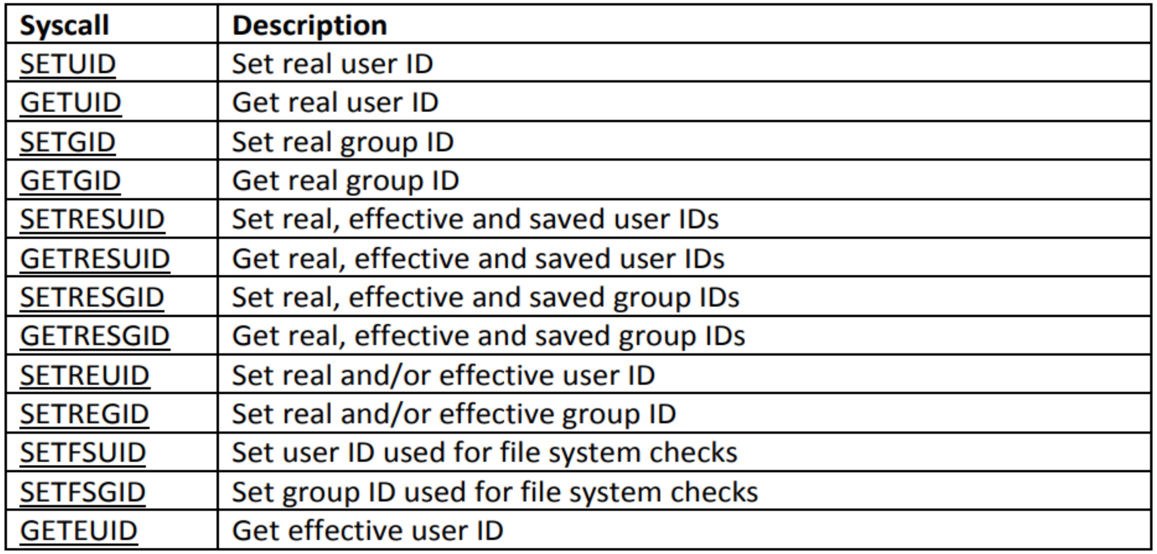


Syscall	Description
SETSID	Set session ID
GETSID	Get session ID

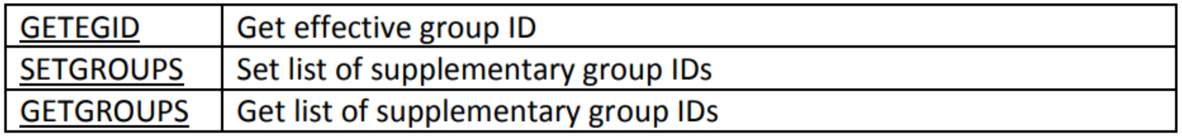
Process Management System Calls in Linux



Syscall	Description
SETPGID	Set process group ID
GETPGID	Get process group ID
GETPGRP	Get the process group ID of the calling process



Syscall	Description
SETUID	Set real user ID
GETUID	Get real user ID
SETGID	Set real group ID
GETGID	Get real group ID
SETRESUID	Set real, effective and saved user IDs
GETRESUID	Get real, effective and saved user IDs
SETRESGID	Set real, effective and saved group IDs
GETRESGID	Get real, effective and saved group IDs
SETREUID	Set real and/or effective user ID
SETREGID	Set real and/or effective group ID
SETFSUID	Set user ID used for file system checks
SETFSGID	Set group ID used for file system checks
GETEUID	Get effective user ID

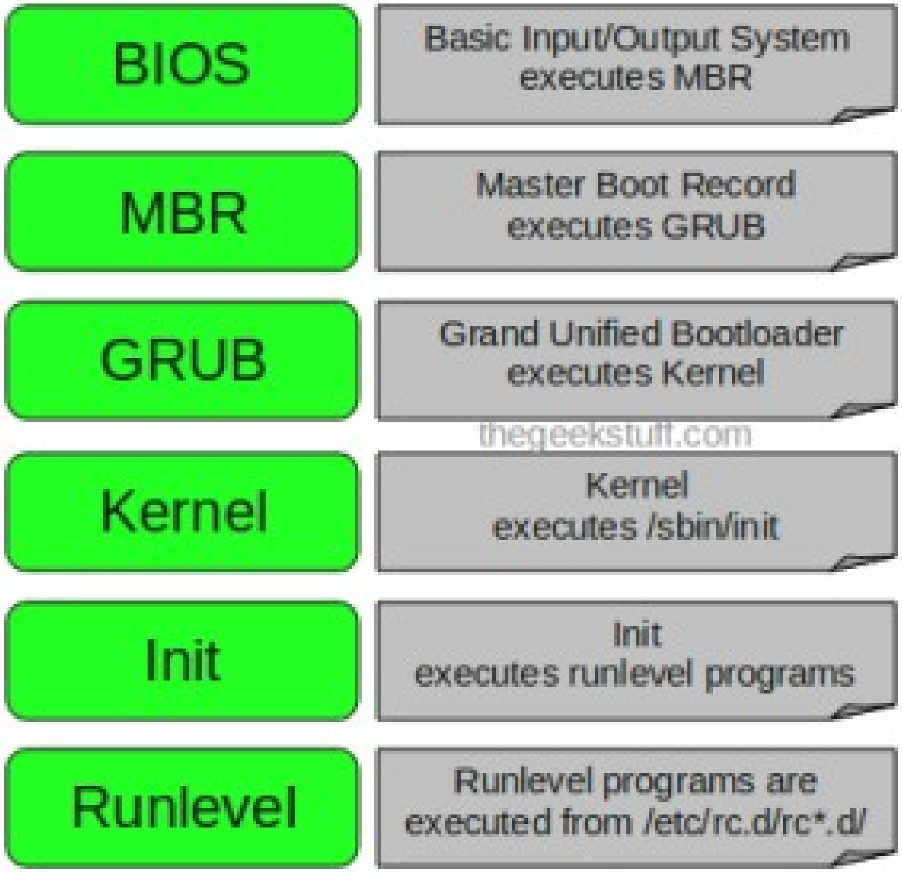


GETEGID	Get effective group ID
SETGROUPS	Set list of supplementary group IDs
GETGROUPS	Get list of supplementary group IDs

User and Kernel Space Thread

- **User Level Threads −** User managed threads.
- **Kernel Level Threads −** Operating System managed threads acting on kernel, an operating system core.
- **User-Space Threads:** cooperative multitasking, user threads typically can switch faster than kernel threads.
- **Kernel-space threads** often are implemented in the kernel using several tables.

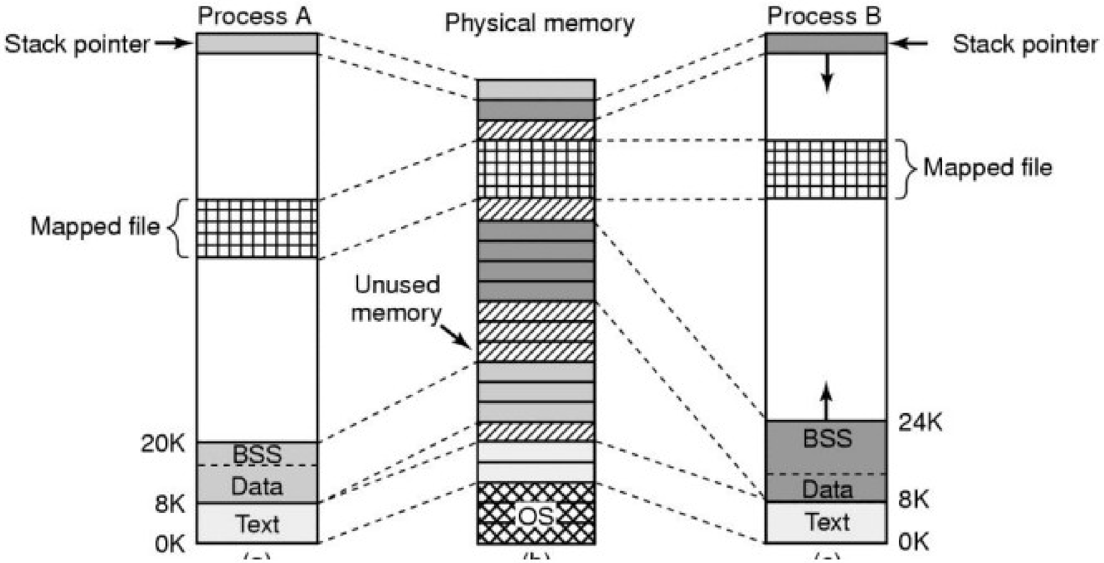
Booting process in Linux



BIOS	Basic Input/Output System executes MBR
MBR	Master Boot Record executes GRUB
GRUB	Grand Unified Bootloader executes Kernel
Kernel	Kernel executes /sbin/init
Init	Init executes runlevel programs
Runlevel	Runlevel programs are executed from /etc/rc.d/rc*.d/

Memory Management in Linux

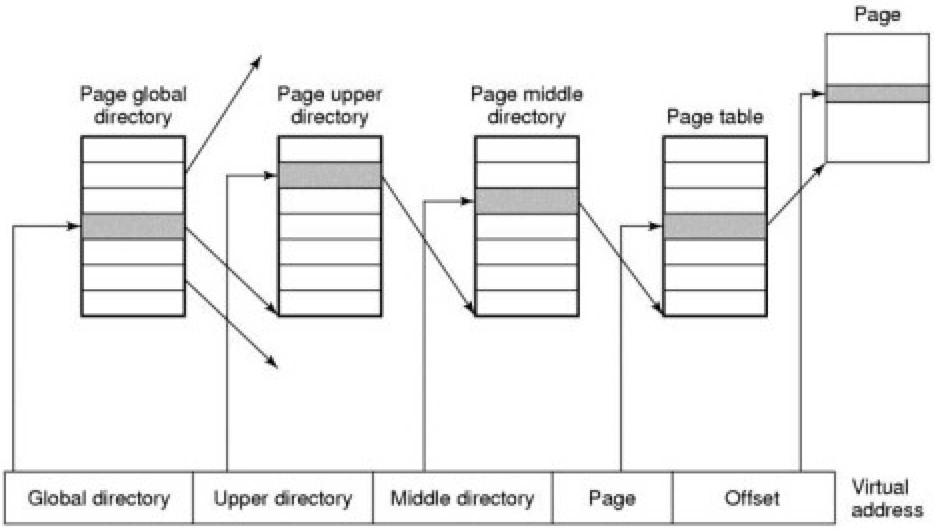
- The memory management subsystem is one of the most important parts of the operating system.
- Virtual memory makes the system appear to have more memory than it actually has by sharing it between competing processes as they need it.


Process A
Stack pointer
Mapped file
20K
BSS
Data
8K
Text
0K
Physical memory
Unused memory
OS
Process B
Stack pointer
Mapped file
24K
BSS
Data
8K
Text
0K

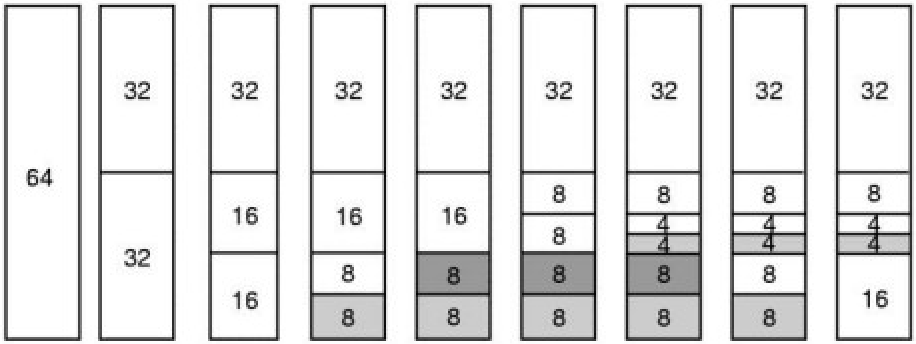
Paging in Linux

- The technique of only loading virtual pages into memory as they are accessed is known as demand paging.
- There are three kernel variables which control the paging operation: minfree, desfree, lotsfree.
- A demand paging system is quite similar to a paging system with swapping where processes reside in secondary memory and pages are loaded only on demand, not in advance.

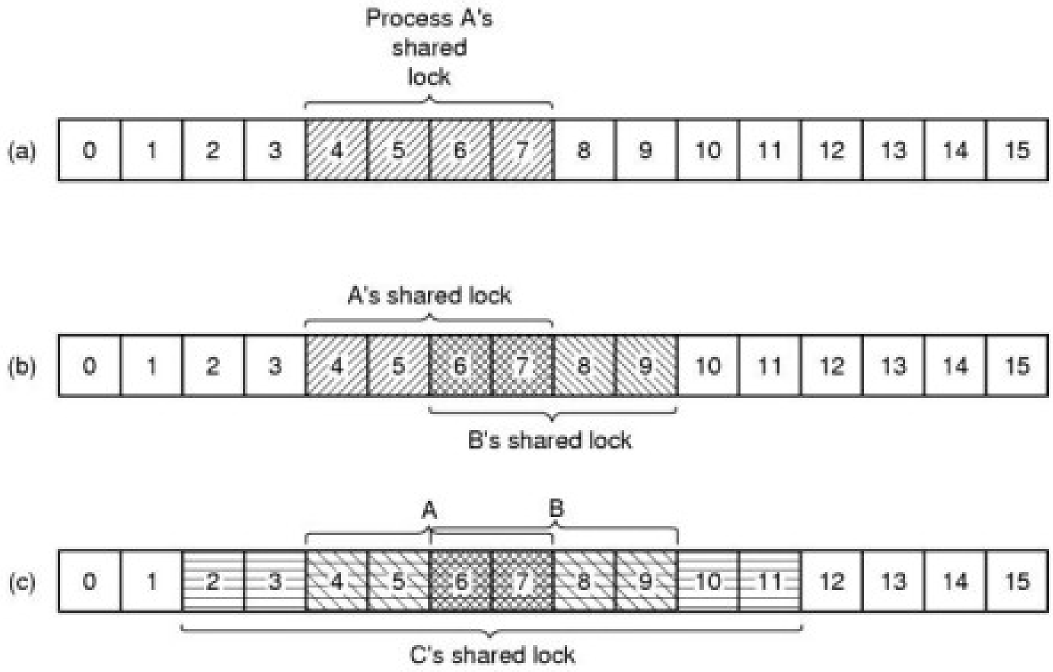
Physical Memory Management


Page global directory
Page upper directory
Page middle directory
Page table
Page
Global directory
Upper directory
Middle directory
Page
Offset
Virtual address

Memory Allocation Mechanisms


64
32 32
32 16 16
32 16 8 8
32 16 8 8
32 8 8 8 8
32 8 4 4 8 8
32 8 4 4 8 8
32 8 4 4 16

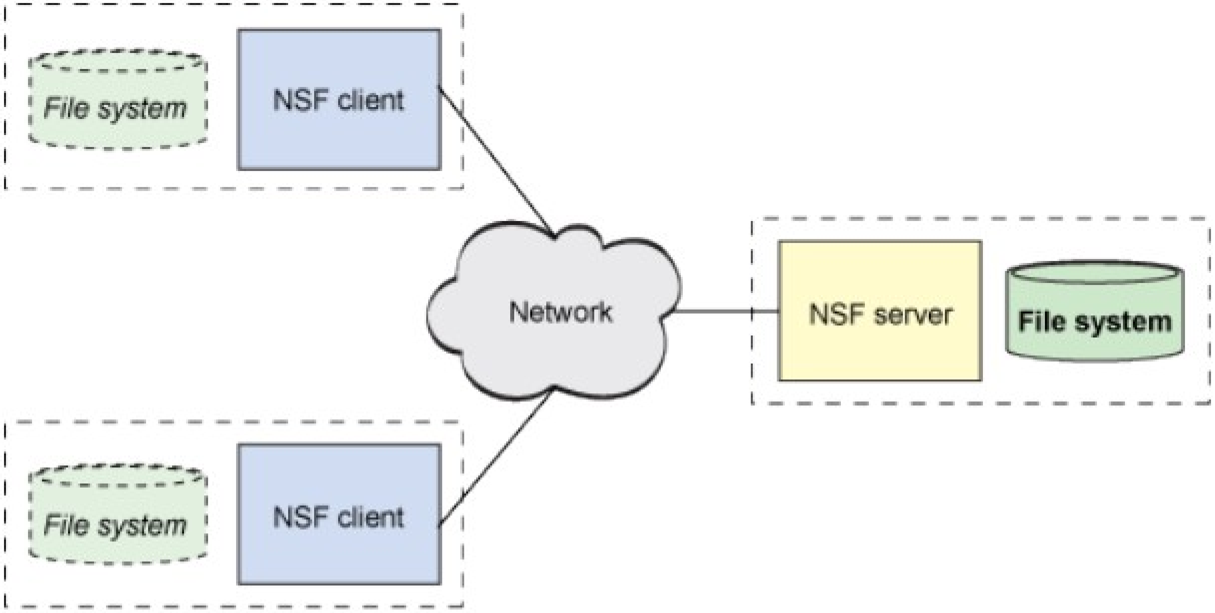
The Linux File System


Process A's shared lock
(a)
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
A's shared lock
(b)
B's shared lock
A
B
(c)
C's shared lock

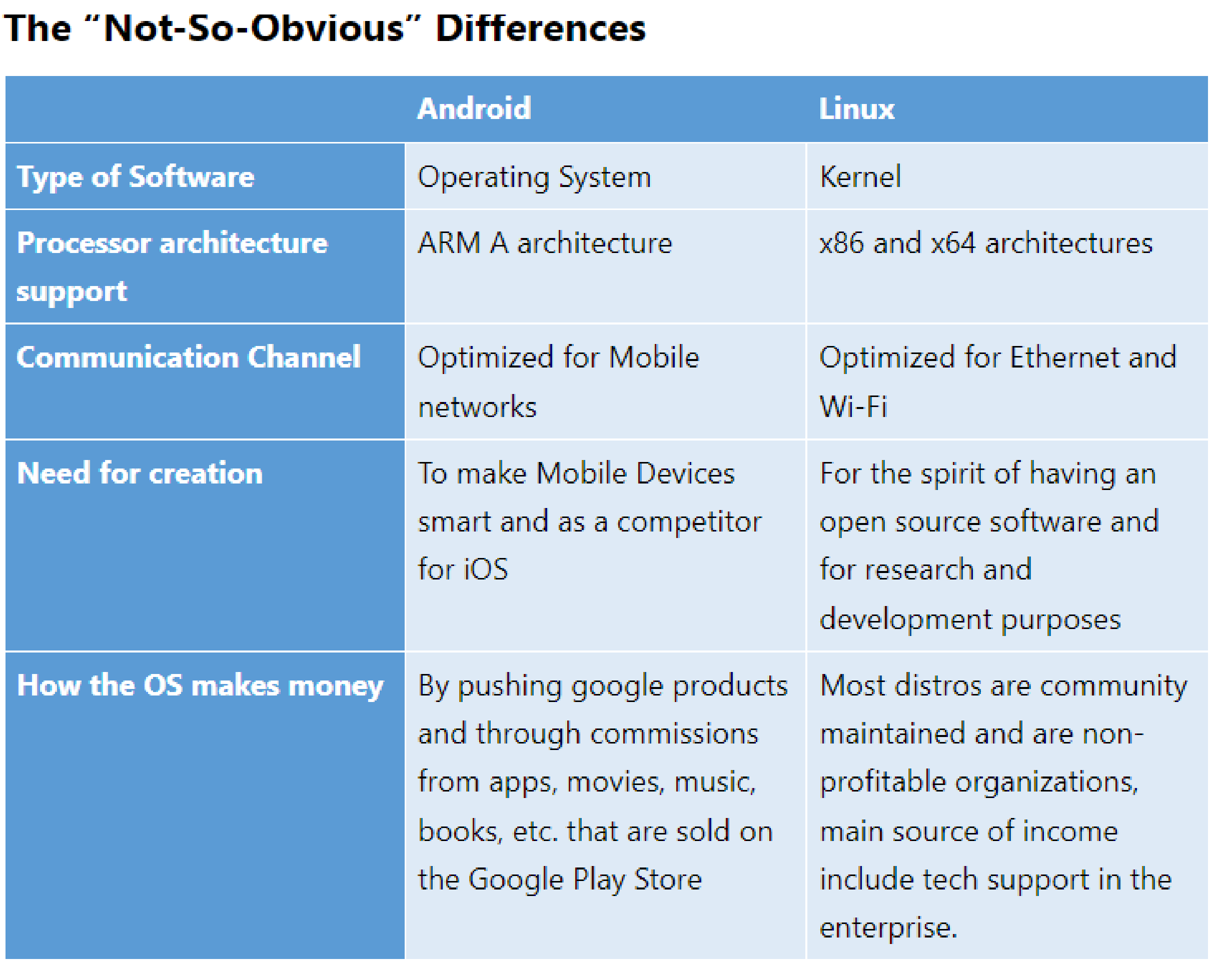
(a) A file with one lock. (b) Addition of a second lock. (c) A third lock.

Network File System (NFS) calls in Linux

- The Network File System (NFS) is a way of mounting Linux discs/directories over a network.
- An NFS server can export one or more directories that can then be mounted on a remote Linux machine.

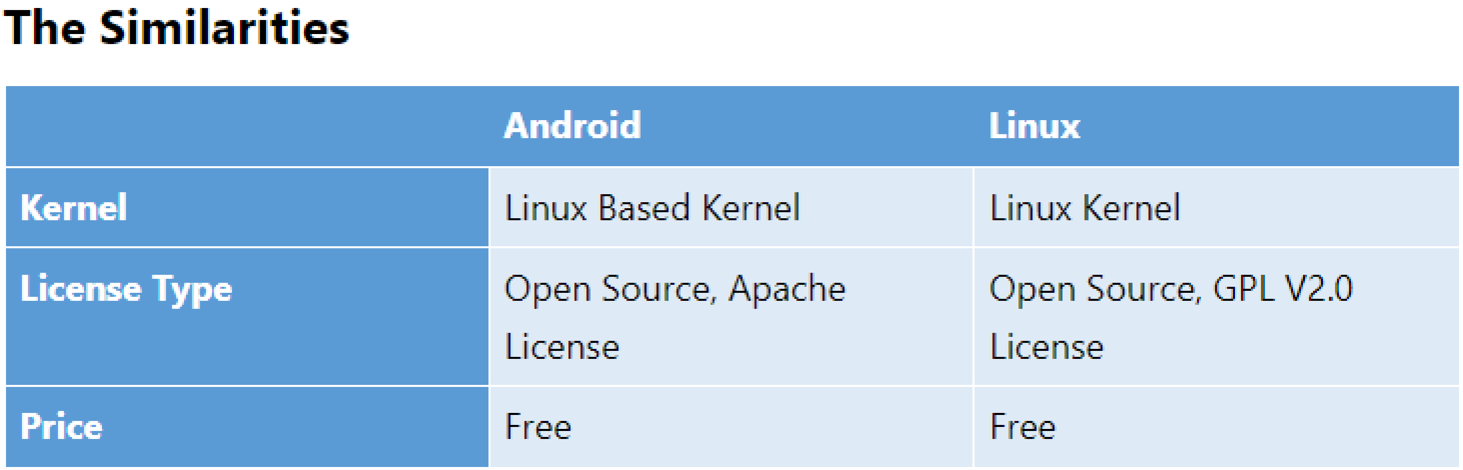

File system
NSF client
Network
NSF server
File system
File system
NSF client

Android vs Linux



The “Not-So-Obvious” Differences

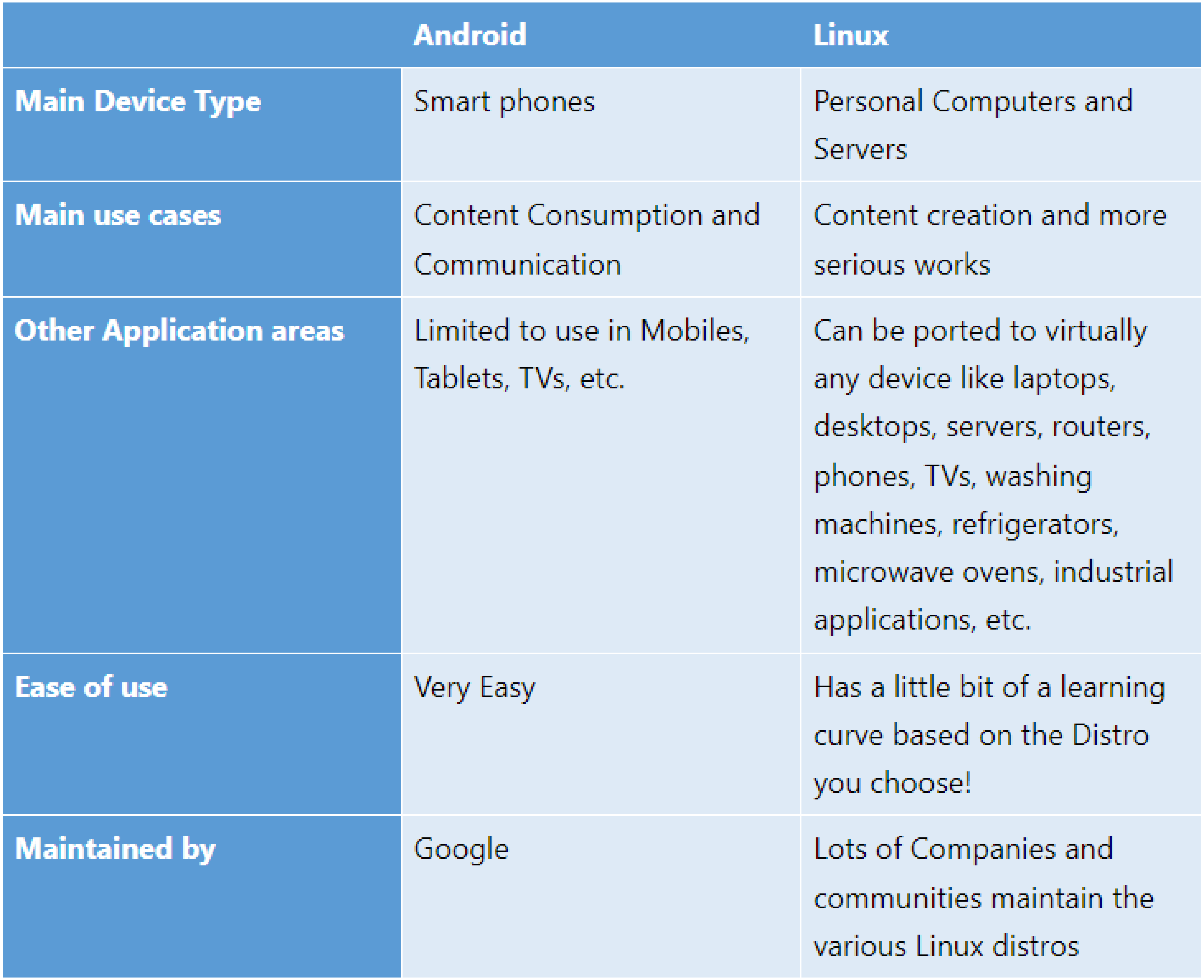
	Android	Linux
Type of Software	Operating System	Kernel
Processor architecture support	ARM A architecture	x86 and x64 architectures
Communication Channel	Optimized for Mobile networks	Optimized for Ethernet and Wi-Fi
Need for creation	To make Mobile Devices smart and as a competitor for iOS	For the spirit of having an open source software and for research and development purposes
How the OS makes money	By pushing google products and through commissions from apps, movies, music, books, etc. that are sold on the Google Play Store	Most distros are community maintained and are non-profitable organizations, main source of income include tech support in the enterprise.



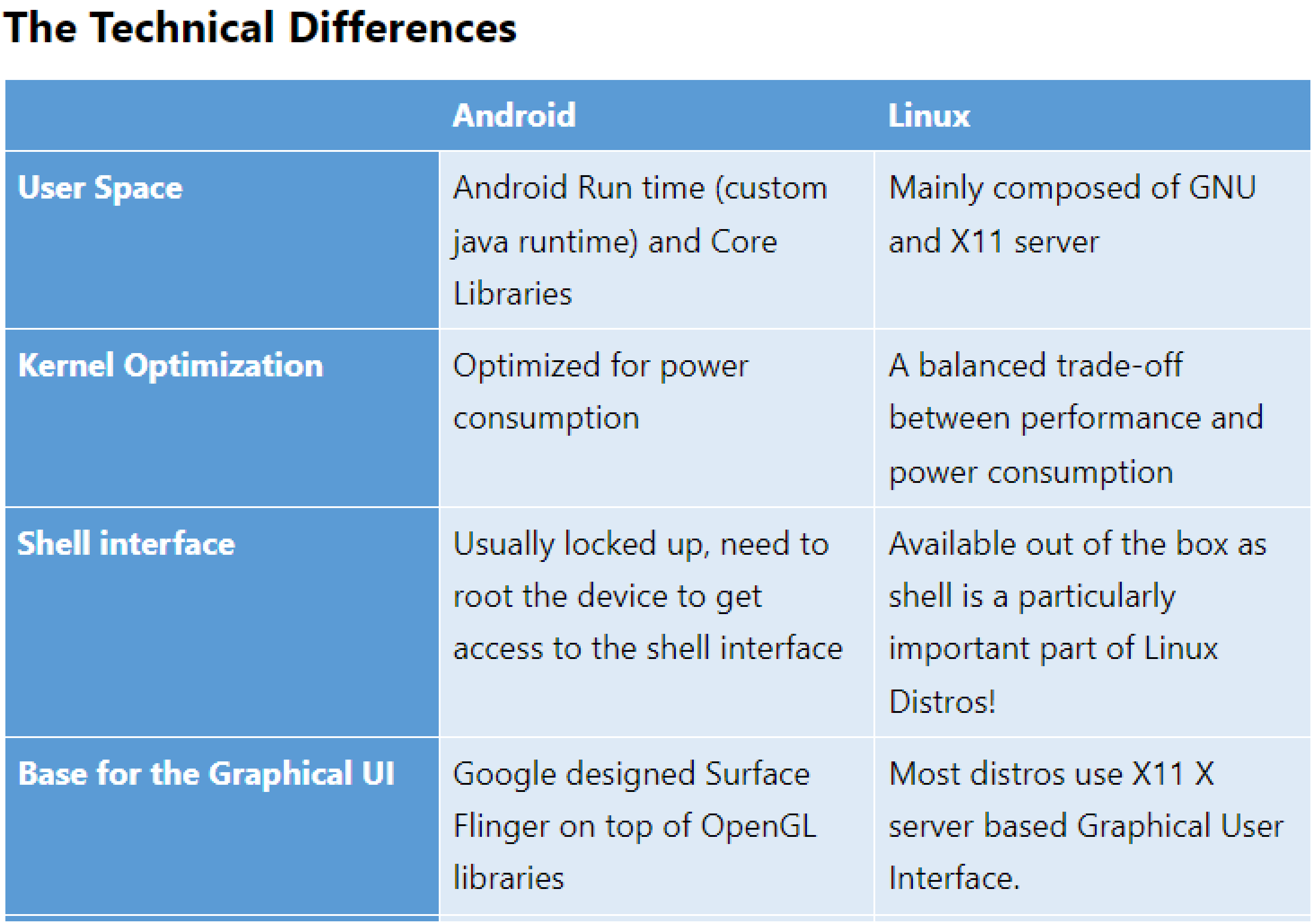
The Similarities

	Android	Linux
Kernel	Linux Based Kernel	Linux Kernel
License Type	Open Source, Apache License	Open Source, GPL V2.0 License
Price	Free	Free

The Obvious Differences



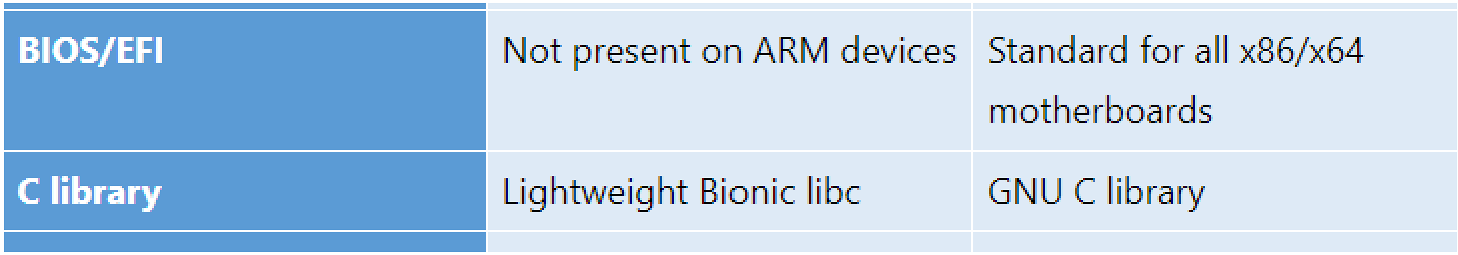
	Android	Linux
Main Device Type	Smart phones	Personal Computers and Servers
Main use cases	Content Consumption and Communication	Content creation and more serious works
Other Application areas	Limited to use in Mobiles, Tablets, TVs, etc.	Can be ported to virtually any device like laptops, desktops, servers, routers, phones, TVs, washing machines, refrigerators, microwave ovens, industrial applications, etc.
Ease of use	Very Easy	Has a little bit of a learning curve based on the Distro you choose!
Maintained by	Google	Lots of Companies and communities maintain the various Linux distros



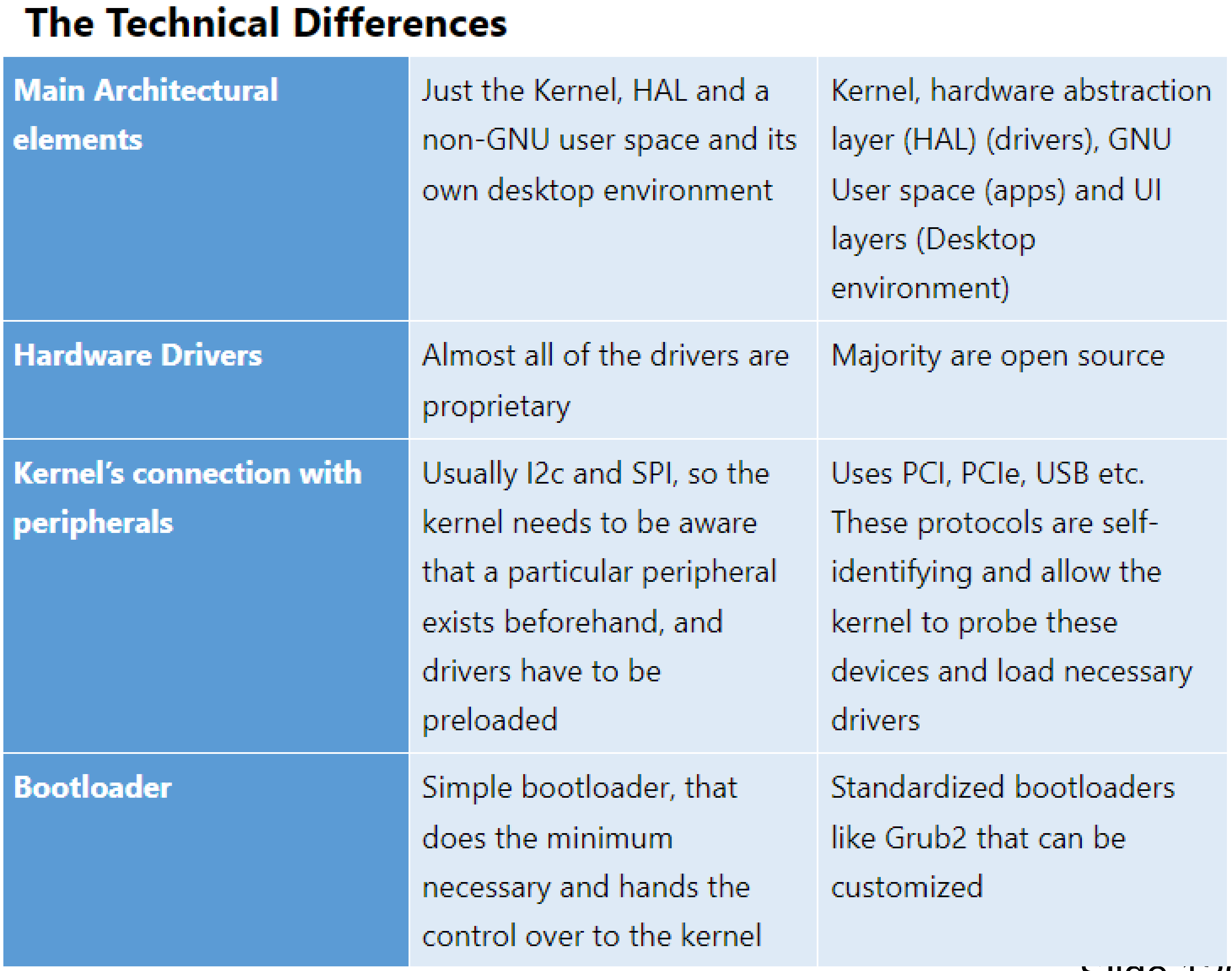
The Technical Differences

	Android	Linux
User Space	Android Run time (custom java runtime) and Core Libraries	Mainly composed of GNU and X11 server
Kernel Optimization	Optimized for power consumption	A balanced trade-off between performance and power consumption
Shell interface	Usually locked up, need to root the device to get access to the shell interface	Available out of the box as shell is a particularly important part of Linux Distros!
Base for the Graphical UI	Google designed Surface Flinger on top of OpenGL libraries	Most distros use X11 X server based Graphical User Interface.

The Technical Differences



BIOS/EFI	Not present on ARM devices	Standard for all x86/x64 motherboards
C library	Lightweight Bionic libc	GNU C library



The Technical Differences

Main Architectural elements	Just the Kernel, HAL and a non-GNU user space and its own desktop environment	Kernel, hardware abstraction layer (HAL) (drivers), GNU User space (apps) and UI layers (Desktop environment)
Hardware Drivers	Almost all of the drivers are proprietary	Majority are open source
Kernel's connection with peripherals	Usually I2c and SPI, so the kernel needs to be aware that a particular peripheral exists beforehand, and drivers have to be preloaded	Uses PCI, PCIe, USB etc. These protocols are self-identifying and allow the kernel to probe these devices and load necessary drivers
Bootloader	Simple bootloader, that does the minimum necessary and hands the control over to the kernel	Standardized bootloaders like Grub2 that can be customized

BITS Pilani
Pilani Campus

THANK YOU

BITS Pilani
Pilani Campus

COMPUTING AND DESIGN

SESSION 6

Course design by - Dr. Lucy J. Gudino &
Dr Chandrasekhar
Faculty – Thangakumar J
WILP & Department of CS & IS

BITS Pilani
Pilani Campus

Operating systems – case study

Android …

- **Android** is a *Linux-based* platform for *mobile devices* …
  - *Operating System*
  - *Middleware*
  - *Applications*
  - *Software Development Kit* (**SDK**)
- Which kind of **mobile devices** … (examples)

SMARTPHONES

TABLETS

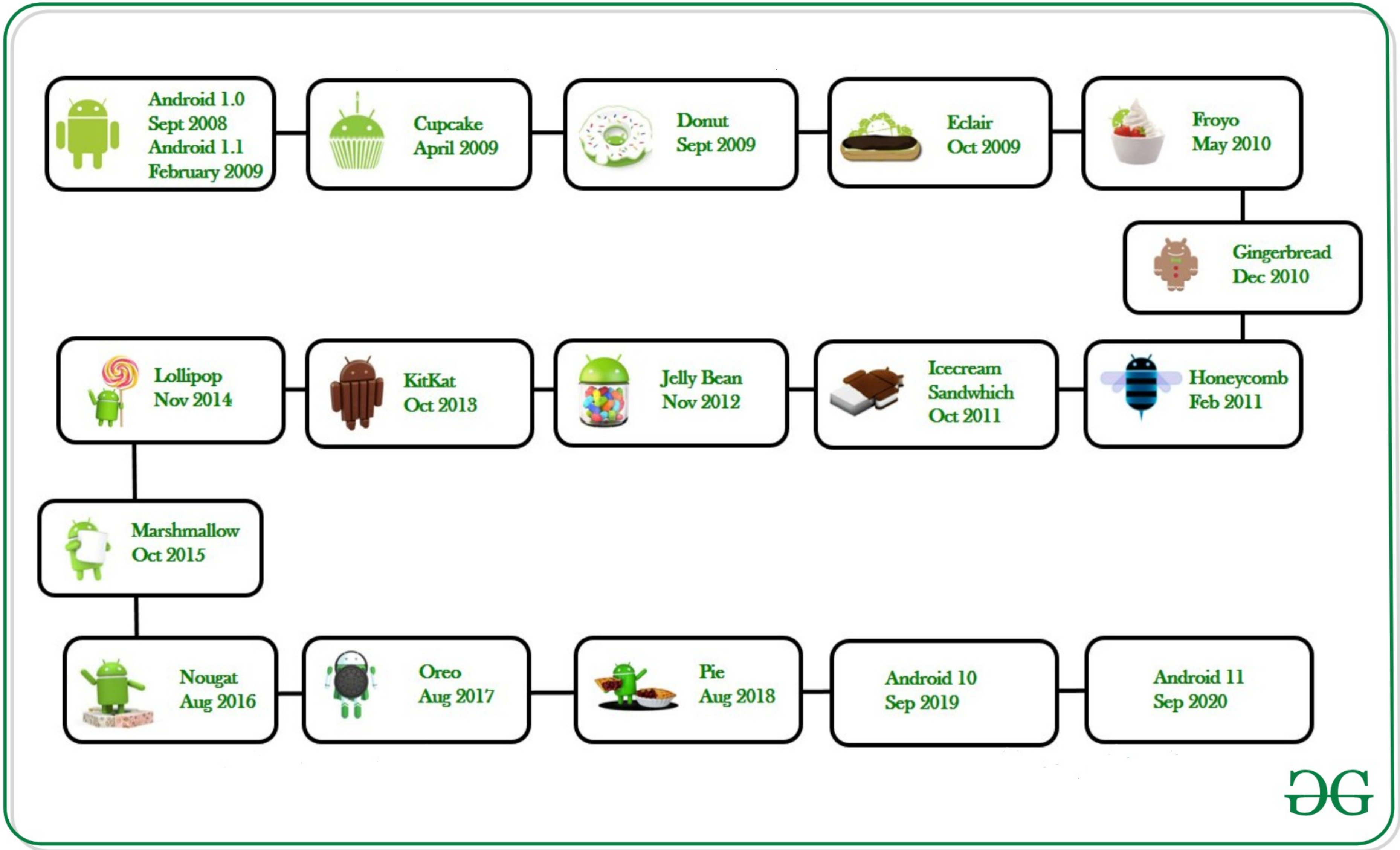
EREADERS

Android TV
HDTV and Internet Video on TV

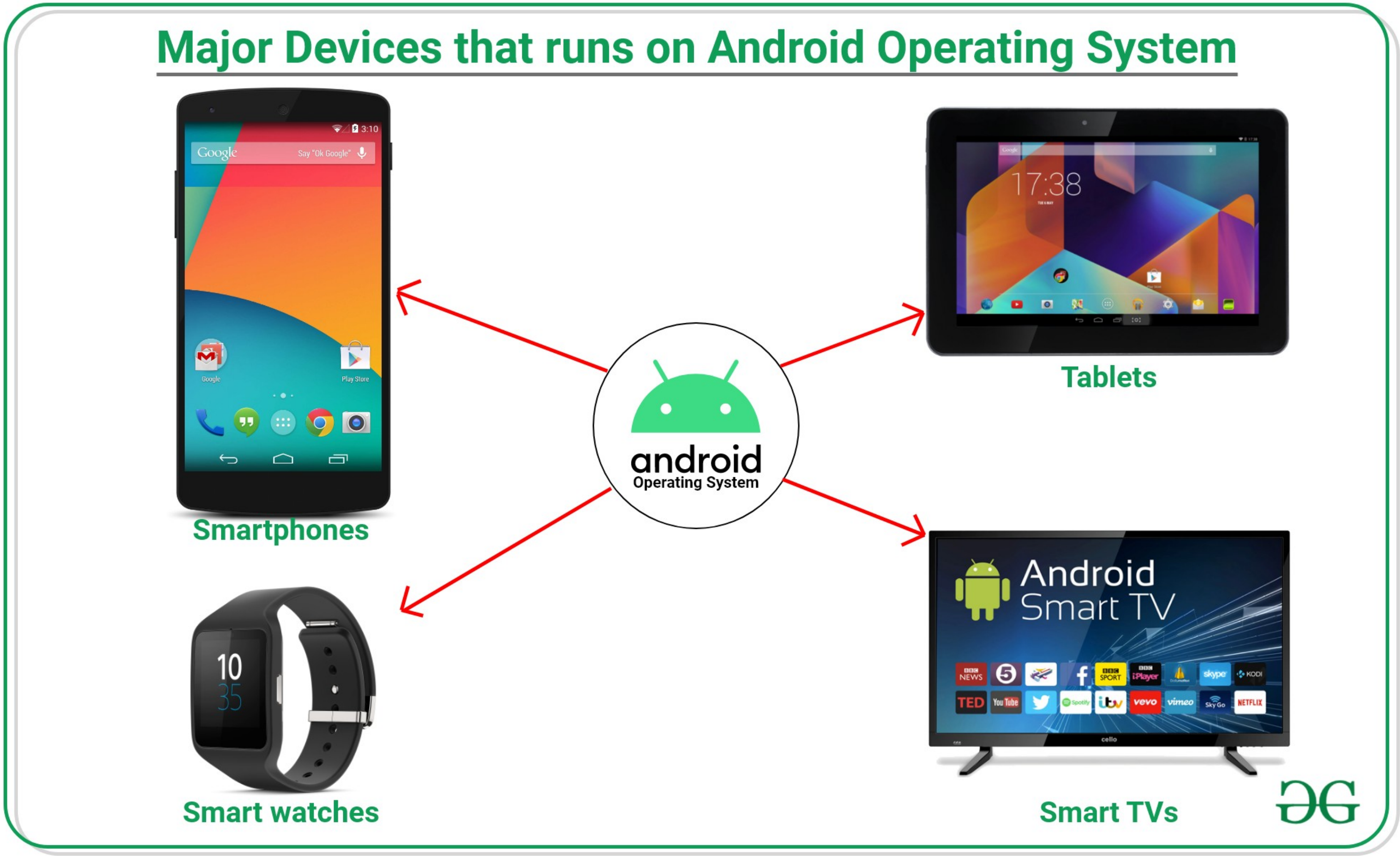
ANDROID TV

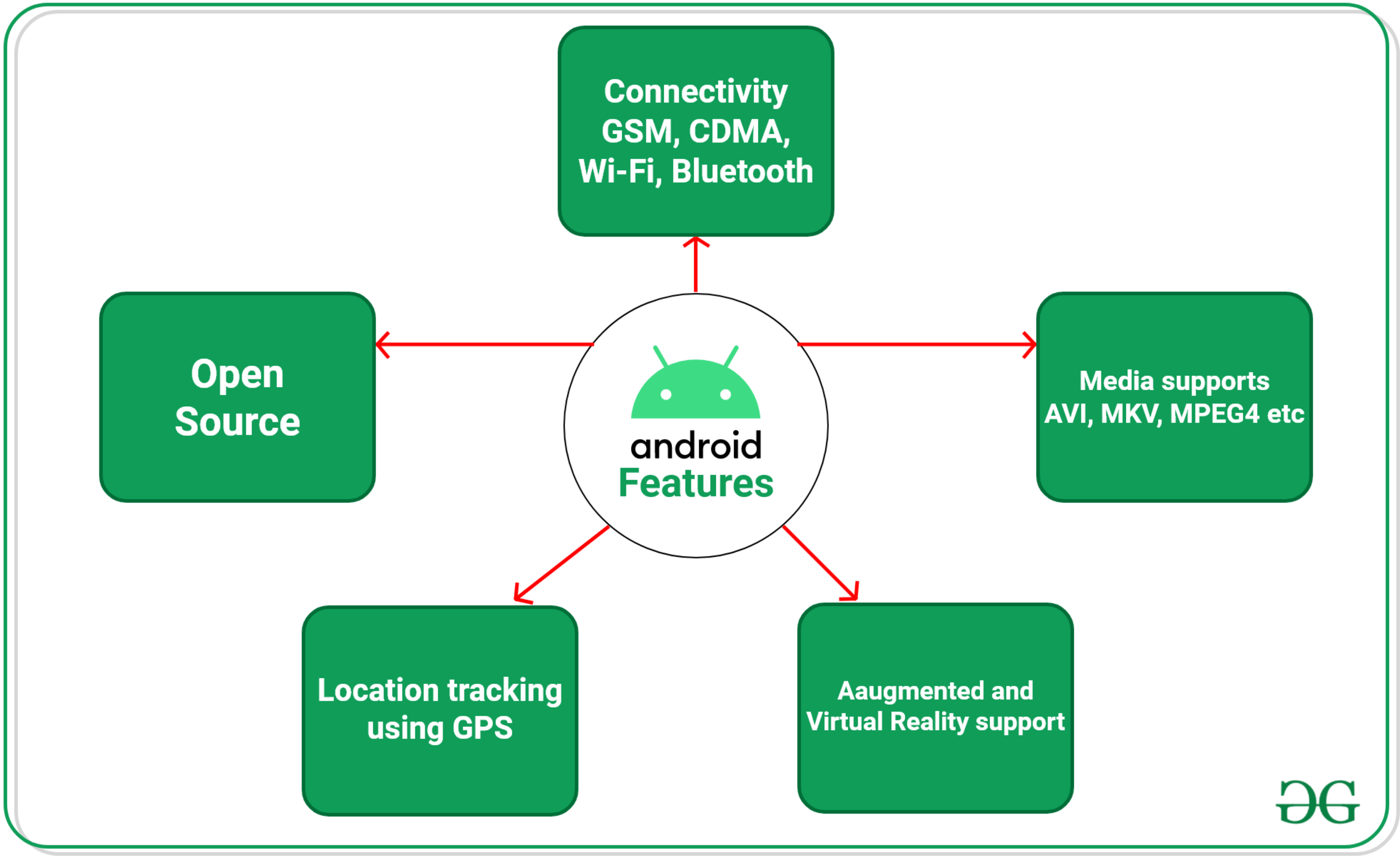
GOOGLE GLASSES

?


Android 1.0
Sept 2008
Android 1.1
February 2009
Cupcake
April 2009
Donut
Sept 2009
Eclair
Oct 2009
Froyo
May 2010
Gingerbread
Dec 2010
Honeycomb
Feb 2011
Icecream
Sandwhich
Oct 2011
Jelly Bean
Nov 2012
KitKat
Oct 2013
Lollipop
Nov 2014
Marshmallow
Oct 2015
Nougat
Aug 2016
Oreo
Aug 2017
Pie
Aug 2018
Android 10
Sep 2019
Android 11
Sep 2020

Versions of Android


Major Devices that runs on Android Operating System
android
Operating System
Smartphones
Tablets
Smart watches
Smart TVs
Android
Smart TV



android Features

Connectivity GSM, CDMA, Wi-Fi, Bluetooth

Open Source

Media supports AVI, MKV, MPEG4 etc

Location tracking using GPS

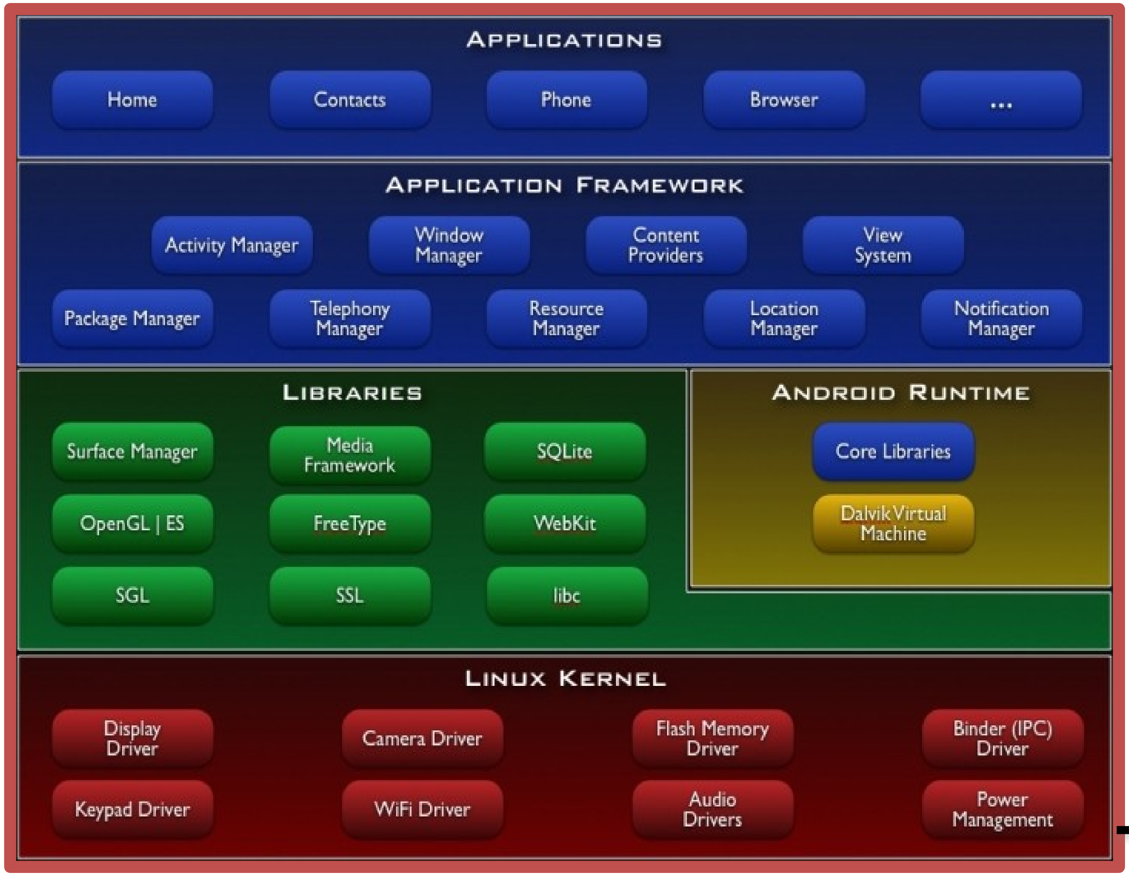
Aaugmented and Virtual Reality support

- Android Open Source Project so we can customize the OS based on our requirements.
- Android supports different types of connectivity for GSM, CDMA, Wi-Fi, Bluetooth, etc. for telephonic conversation or data transfer.
- Using wifi technology we can pair with other devices while playing games or using other applications.
- It contains multiple APIs to support location-tracking services such as GPS.
- We can manage all data storage-related activities by using the file manager.

- It contains a wide range of media supports like AVI, MKV, FLV, MPEG4, etc. to play or record a variety of audio/video.
- It also supports different image formats like JPEG, PNG, GIF, BMP, MP3, etc.
- It supports multimedia hardware control to perform playback or recording using a camera and microphone.

- Android has an integrated open-source WebKit layout-based web browser to support User Interfaces like HTML5, and CSS3.
- Android supports multi-tasking means we can run multiple applications at a time and can switch between them.
- It provides support for virtual reality or 2D/3D Graphics.

The Android Architecture


Applications
Home
Contacts
Phone
Browser
...
Application Framework
Activity Manager
Window Manager
Content Providers
View System
Package Manager
Telephony Manager
Resource Manager
Location Manager
Notification Manager
Libraries
Surface Manager
Media Framework
SQLite
OpenGL | ES
FreeType
WebKit
SGL
SSL
libc
Android Runtime
Core Libraries
Dalvik Virtual Machine
Linux Kernel
Display Driver
Camera Driver
Flash Memory Driver
Binder (IPC) Driver
Keypad Driver
WiFi Driver
Audio Drivers
Power Management

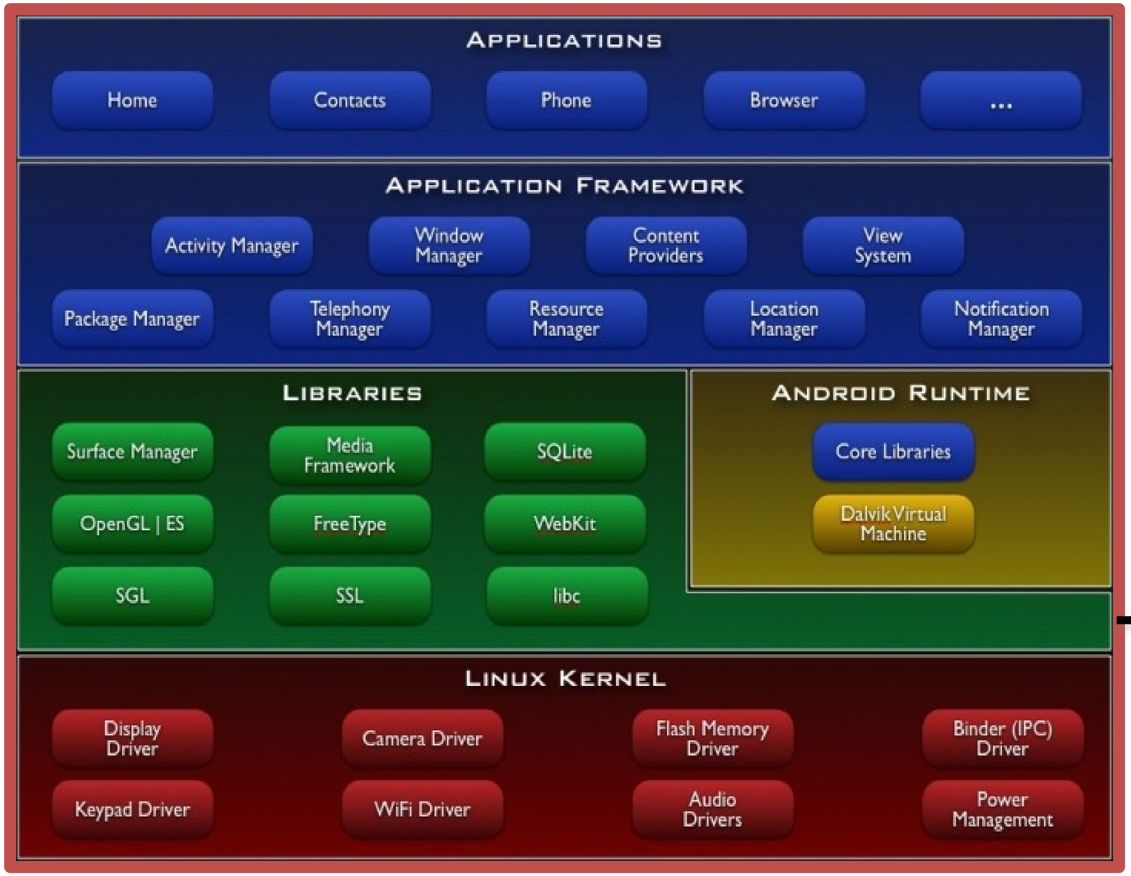
Built on top of **Linux kernel**

Advantages:

- **Portability** (i.e. easy to compile on different harwdare architectures)
- **Security** (e.g. secure multi-process environment)
- **Power** Management

• Memory and Process management ▫ File & Network I/O ▫ Device Drivers • Preemptive Multitasking • Lean, efficient, and secure • Open Source

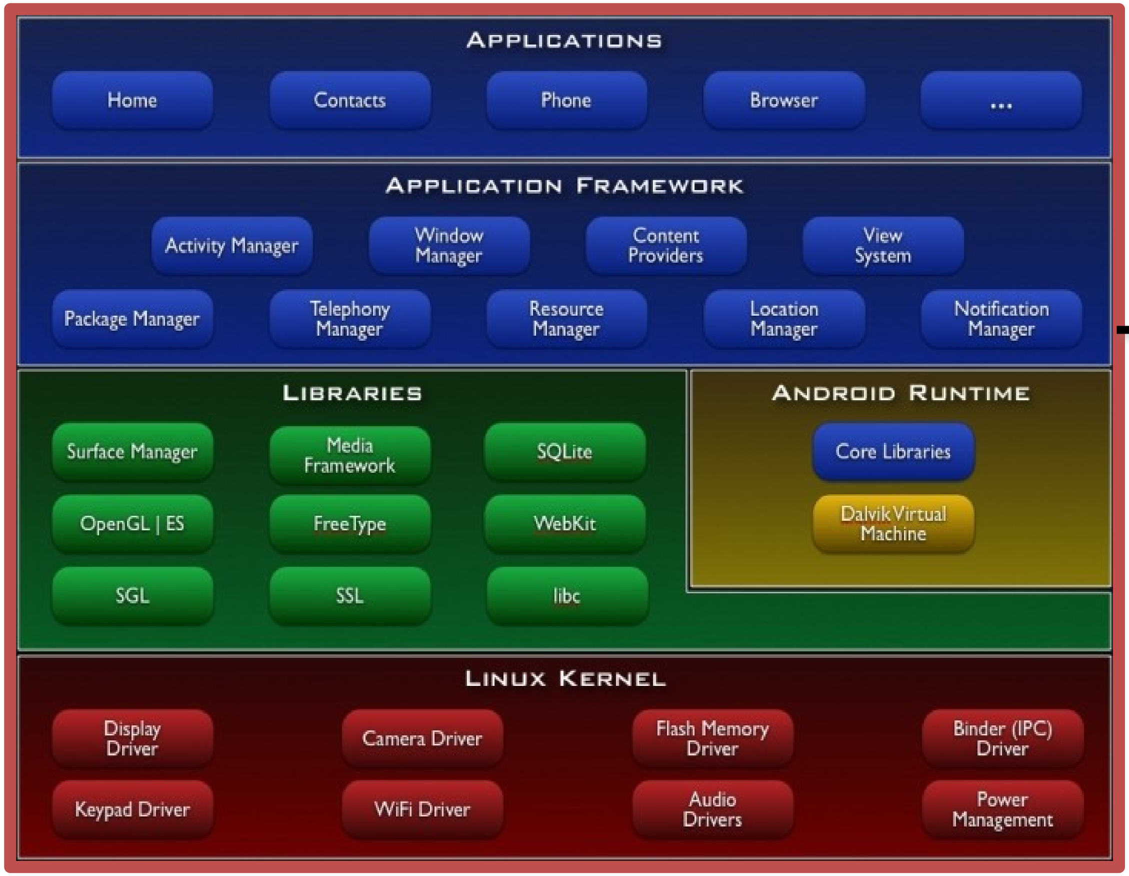
The Android Architecture


APPLICATIONS
Home
Contacts
Phone
Browser
...
APPLICATION FRAMEWORK
Activity Manager
Window Manager
Content Providers
View System
Package Manager
Telephony Manager
Resource Manager
Location Manager
Notification Manager
LIBRARIES
Surface Manager
Media Framework
SQLite
OpenGL | ES
FreeType
WebKit
SGL
SSL
libc
ANDROID RUNTIME
Core Libraries
Dalvik Virtual Machine
LINUX KERNEL
Display Driver
Camera Driver
Flash Memory Driver
Binder (IPC) Driver
Keypad Driver
WiFi Driver
Audio Drivers
Power Management

Native **Libraries**
(C/C++ code)

- **Graphics** (Surface Manager)
- **Multimedia** (Media Framework)
- **Database DBMS** (SQLite)
- **Font Management** (FreeType)
- **WebKit**
- **C libraries** (Bionic)
- **….**

The Android Architecture

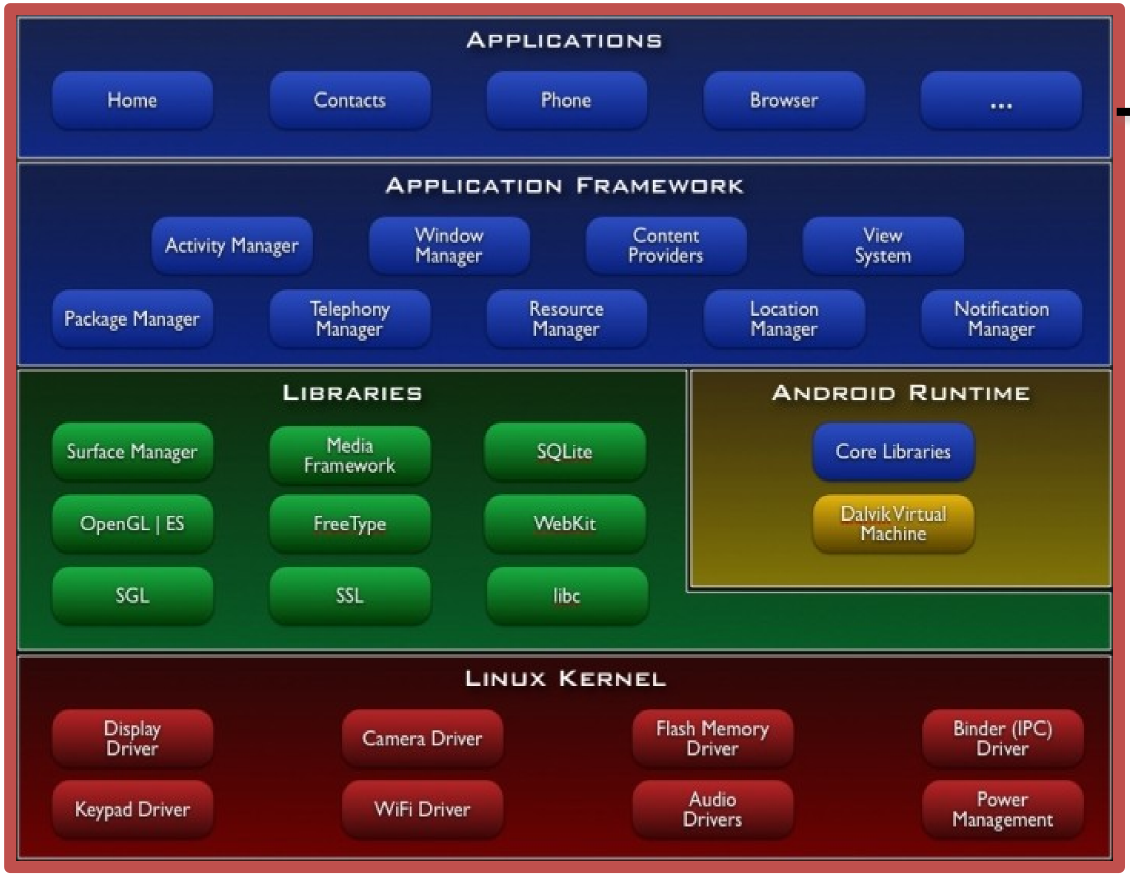

APPLICATIONS
Home
Contacts
Phone
Browser
...
APPLICATION FRAMEWORK
Activity Manager
Window Manager
Content Providers
View System
Package Manager
Telephony Manager
Resource Manager
Location Manager
Notification Manager
LIBRARIES
Surface Manager
Media Framework
SQLite
OpenGL | ES
FreeType
WebKit
SGL
SSL
libc
ANDROID RUNTIME
Core Libraries
Dalvik Virtual Machine
LINUX KERNEL
Display Driver
Camera Driver
Flash Memory Driver
Binder (IPC) Driver
Keypad Driver
WiFi Driver
Audio Drivers
Power Management

Application Libraries

(Core Components of Android)

- **Activity Manager**
- **Packet Manager**
- **Telephony Manager**
- **Location Manager**
- **Contents Provider**
- **Notification Manager**
- **….**

The Android Architecture

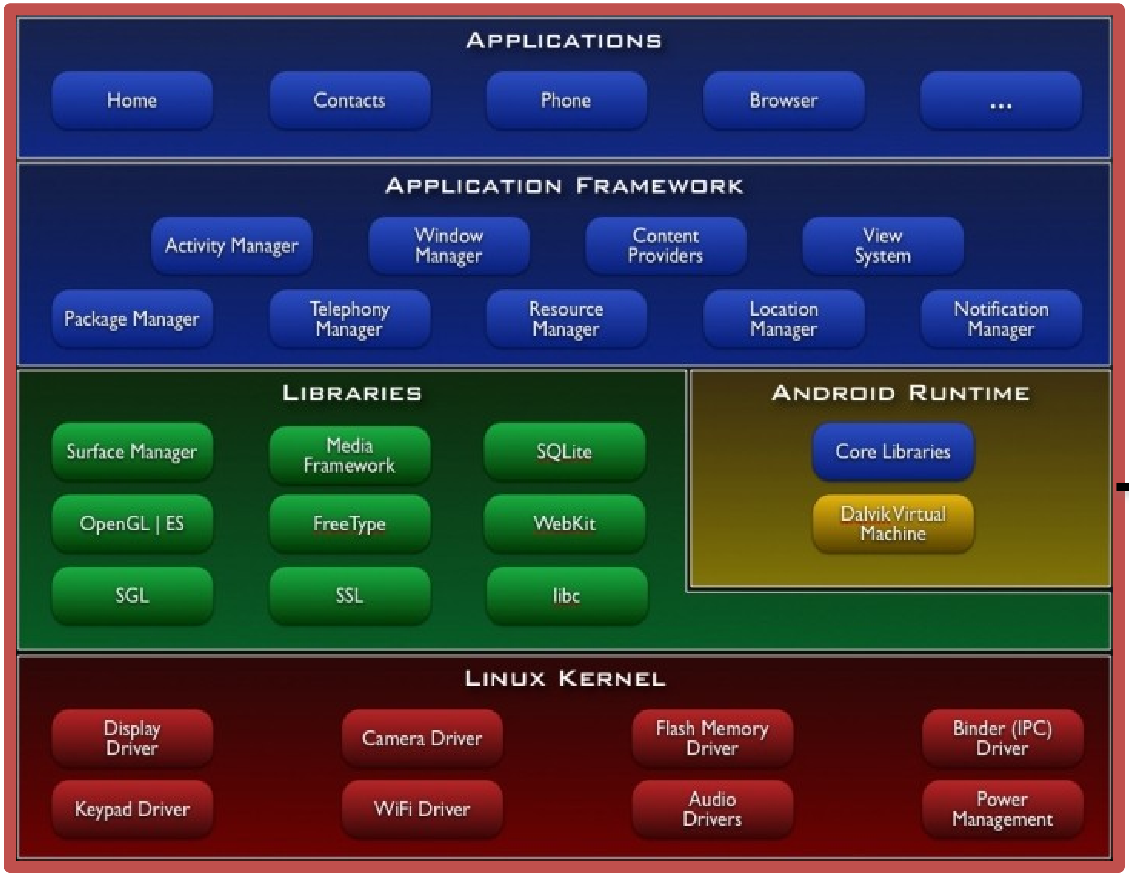

APPLICATIONS
Home
Contacts
Phone
Browser
...
APPLICATION FRAMEWORK
Activity Manager
Window Manager
Content Providers
View System
Package Manager
Telephony Manager
Resource Manager
Location Manager
Notification Manager
LIBRARIES
Surface Manager
Media Framework
SQLite
OpenGL | ES
FreeType
WebKit
SGL
SSL
libc
ANDROID RUNTIME
Core Libraries
Dalvik Virtual Machine
LINUX KERNEL
Display Driver
Camera Driver
Flash Memory Driver
Binder (IPC) Driver
Keypad Driver
WiFi Driver
Audio Drivers
Power Management

Applications

(Written in **Java** code)

- **Android Play Store**
- **Entertainment**
- **Productivity**
- **Personalization**
- **Education**
- **Geo-communication**
- **….**

The Android Architecture


APPLICATIONS
Home
Contacts
Phone
Browser
...
APPLICATION FRAMEWORK
Activity Manager
Window Manager
Content Providers
View System
Package Manager
Telephony Manager
Resource Manager
Location Manager
Notification Manager
LIBRARIES
Surface Manager
Media Framework
SQLite
OpenGL | ES
FreeType
WebKit
SGL
SSL
libc
ANDROID RUNTIME
Core Libraries
Dalvik Virtual Machine
LINUX KERNEL
Display Driver
Camera Driver
Flash Memory Driver
Binder (IPC) Driver
Keypad Driver
WiFi Driver
Audio Drivers
Power Management

Dalvik Virtual Machine (VM)

- **Novel** Java Virtual Machine implementation (not using the Oracle JVM)
- Open **License** (Oracle JVM is not open!)
- **Optimized** for memory-constrained devices
- **Faster** than Oracle JVM
- ….

Android Applications Design

APPLICATION **DESIGN**:

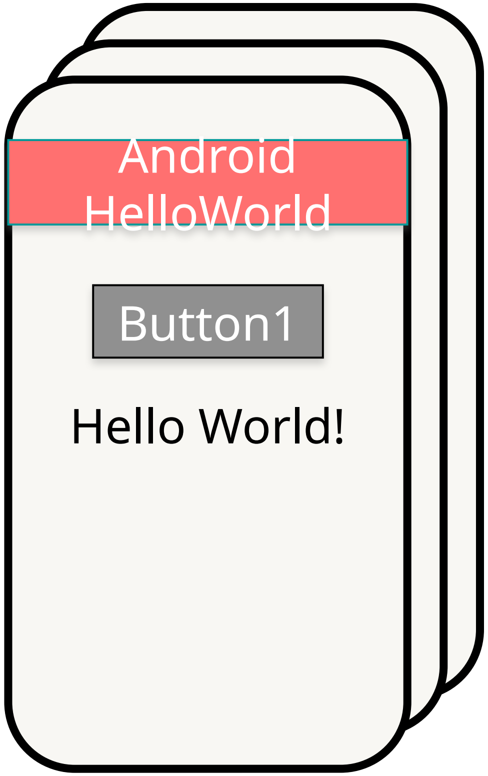
- **GUI** Definition
- **Events** Management
- Application **Data** Management
- **Background** Operations
- **User** Notifications

Android Applications Design

APPLICATION **COMPONENTS**

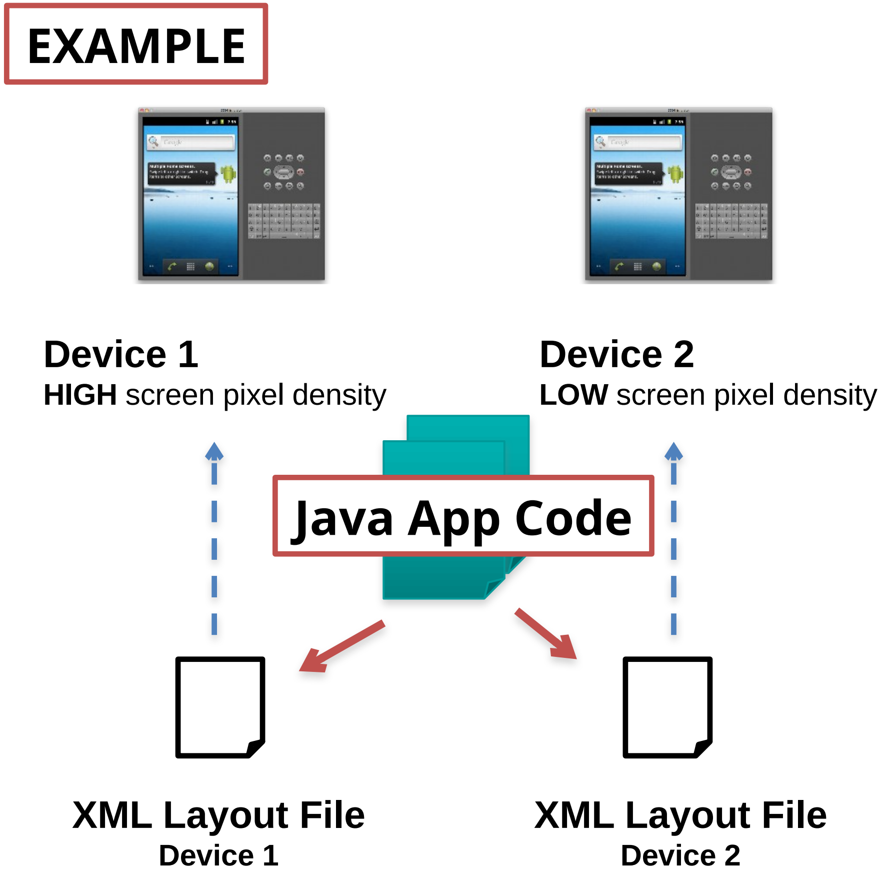
- **Activities**
- **Intents**
- **Services**
- **Content Providers**
- **Broadcast Receivers**

Android Components: Activities


Android HelloWorld
Button1
Hello World!

- An **Activity** corresponds to a **single screen** of the **Application**.
- An Application can be composed of *multiples screens* (Activities).
- The **Home Activity** is shown when the user launches an application.
- Different activities can exhange information one with each other.

Android Components: Activities



EXAMPLE

Device 1
HIGH screen pixel density

Device 2
LOW screen pixel density

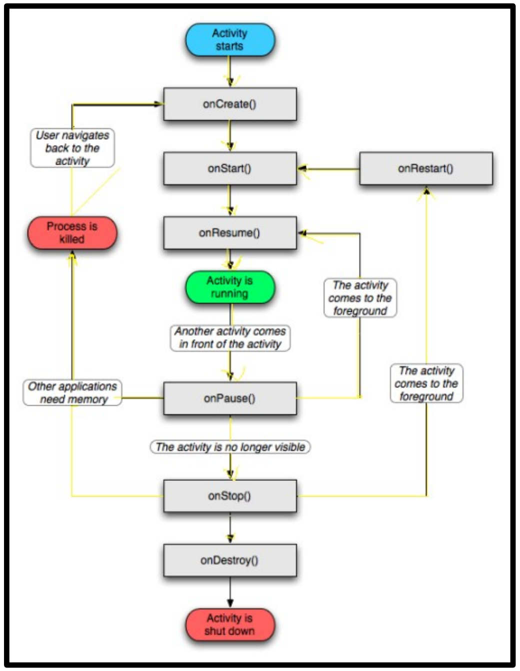
Java App Code

XML Layout File
Device 1

XML Layout File
Device 2

- Build the **application layout** through XML files (like HTML)
- Define **two** different XML **layouts** for two different devices
- At **runtime**, Android detects the current device configuration and loads the appropriate resources for the application
- **No need to recompile**!
- Just add a new XML file if you need to support a new device

Android Components: Activities


Activity starts
onCreate()
User navigates back to the activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
The activity comes to the foreground
Another activity comes in front of the activity
Other applications need memory
onPause()
The activity comes to the foreground
The activity is no longer visible
onStop()
onDestroy()
Activity is shut down

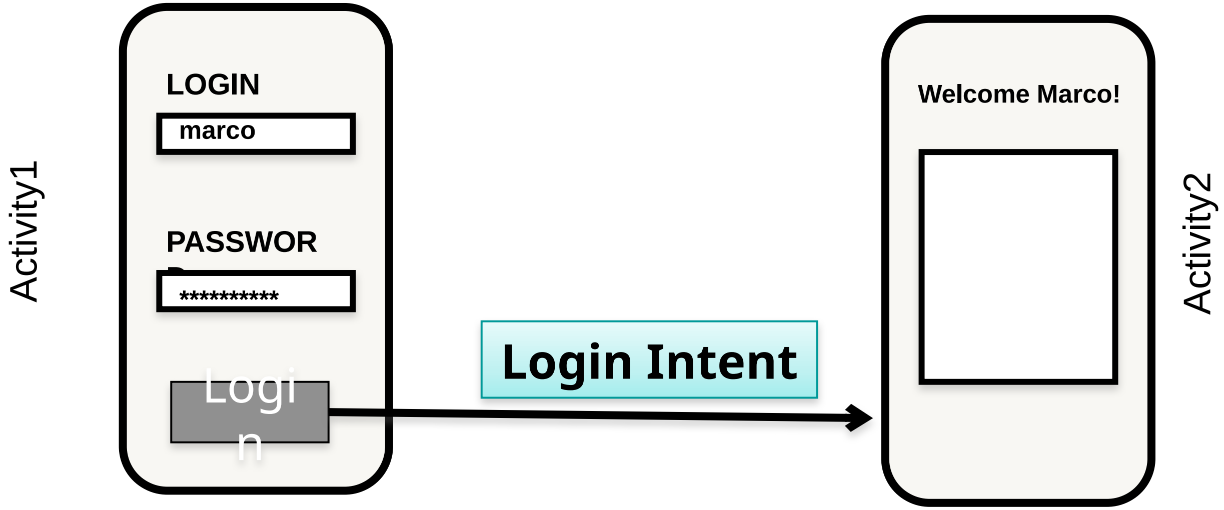
- The **Activity Manager** is responsible for creating, destroying, managing activities.
- Activities can be on different **states**: *starting*, *running*, *stopped*, *destroyed*, *paused*.
- Only one activity can be on the **running** state at a time.
- Activities are organized on a **stack**, and have an event-driven life cycle (details later …)

Android Components: Activities

- Main difference between Android-programming and Java (Oracle) -programming:
  - **Mobile devices have constrained resource capabilities!**
- Activity lifetime depends on **users' choice** (i.e. change of visibility) as well as on **system contraints** (i.e. memory shortage).
- Developer must implement **lifecycle methods** to account for state changes of each Activity …

Android Components: Intents

- **Intents**: asynchronous **messages** to activate core Android components (e.g. Activities).
- **Explicit** Intent → The component *(e.g. Activity1)* specifies the destination of the intent *(e.g. Activity 2)*.



Activity1

LOGIN

marco

PASSWOR

****

Login

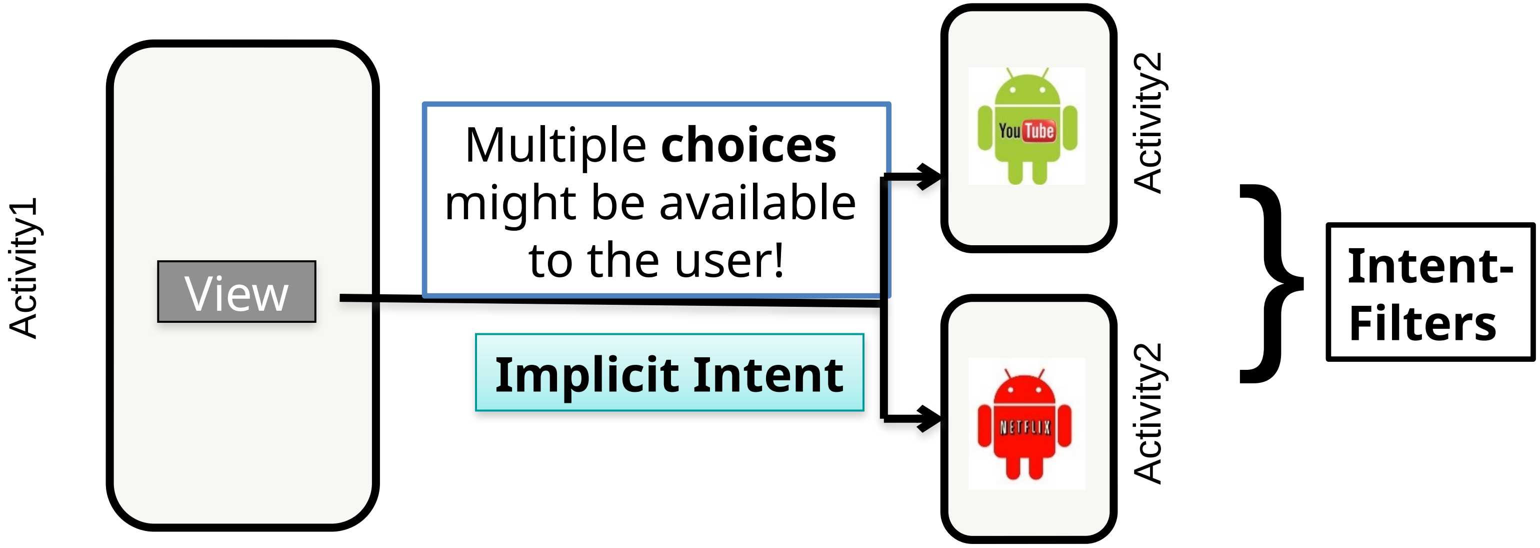
Login Intent

Welcome Marco!

Activity2

Android Components: Intents

- **Intents**: asynchronous **messages** to activate core Android components (e.g. Activities).
- **Implicit** Intent → The component *(e.g. Activity1)* specifies the type of the intent *(e.g. "View a video")*.


Activity1
View
Activity2
Activity2
You Tube
NETFLIX

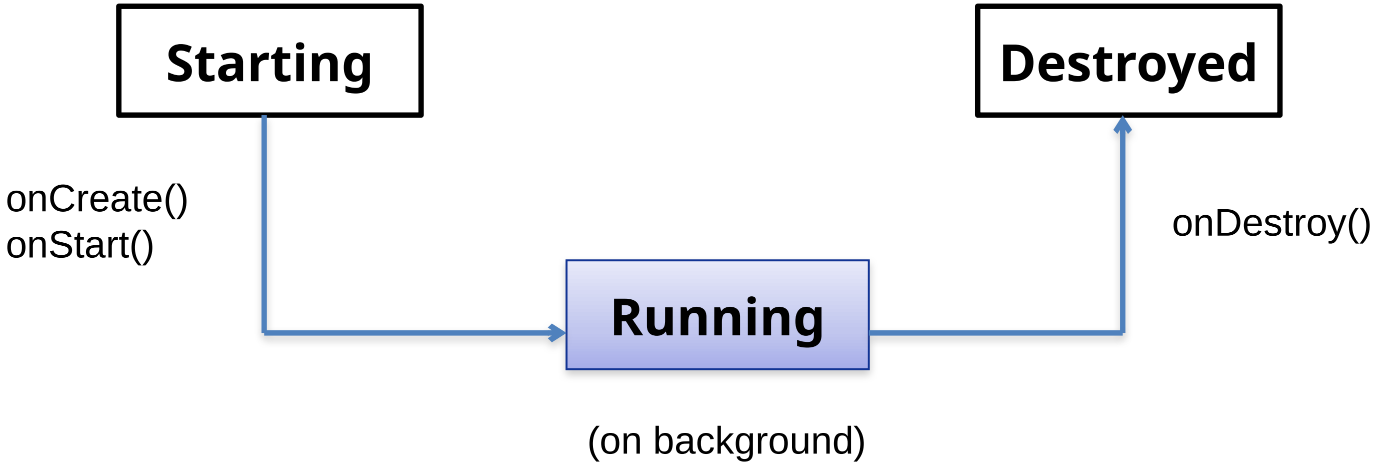
Multiple **choices** might be available to the user!

Implicit Intent

Intent-Filters

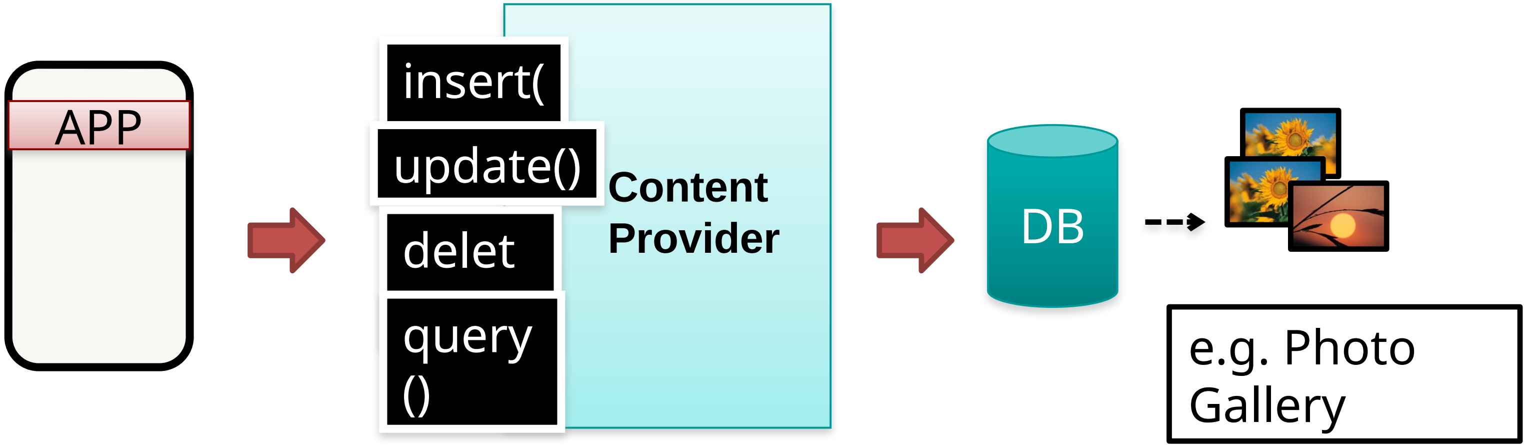
Android Components: Services

- **Services**: like Activities, but run in **background** and do not provide an user interface.
- Used for **non-interactive** tasks (e.g. networking).
- Service life-time composed of 3 states:


Starting
Destroyed
onCreate()
onStart()
onDestroy()
Running
(on background)

Android Components: Content Providers

- Each Android **application** has its own **private** set of data (managed through *files* or through *SQLite* database).
- **Content Providers**: Standard **interface** to *access and share data among different applications*.



APP

insert(

update()

delet

query ()

Content Provider

DB

e.g. Photo Gallery

Android Components: System API

- Using the **components** described so far, Android applications can then leverage the system API …

SOME EXAMPLEs …

- *Telephony Manager* data access (call, SMS, etc)
- *Sensor* management (GPS, accelerometer, etc)
- Network *connectivity* (Wifi, bluetooth, NFC, etc)
- *Web* surfing (HTTP client, WebView, etc)
- *Storage* management (files, SQLite db, etc)
- ….

Android Components: Google API

- … or easily interface with other **Google services**:

Google
maps

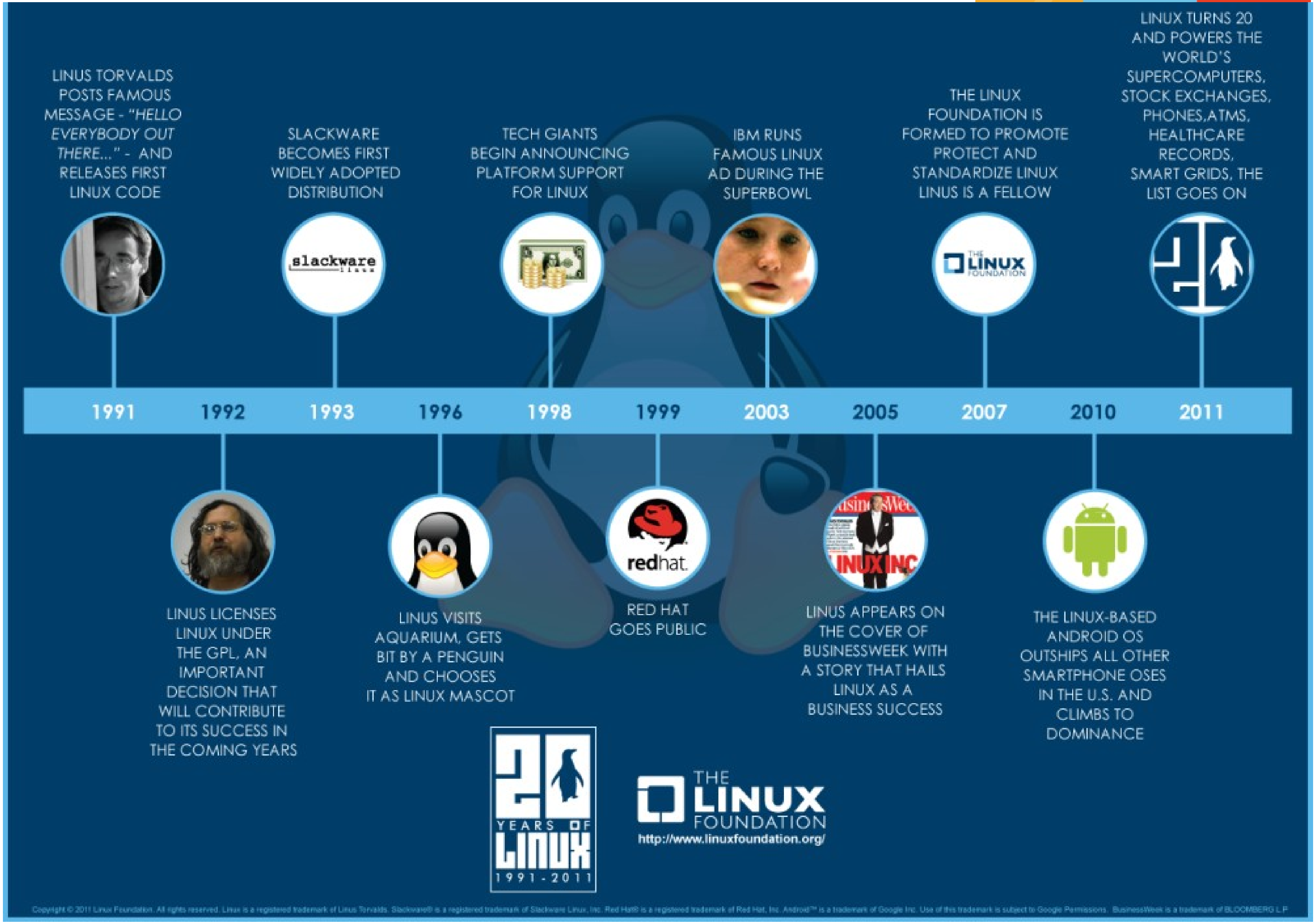
Google
places

Android Application Security

- Android applications run with a distinct system identity (Linux user ID and group ID), in an **isolated** way.
- Applications must explicitly share resources and data. They do this by declaring the ***permissions*** they need for additional capabilities.
  - Applications statically **declare** the permissions they require.
  - User must **give his/her consensus** during the installation.

Case Study - Linux

1. **Evolution of UNIX**
2. **LINUX Operating System & Design goals**
3. **Interfaces to Linux**
4. **List of Linux Utility Programs**
5. **Linux kernel**
6. **Process in Linux and PID, UID, GID in Linux**
7. **Process Management System Calls in Linux**
8. **User space thread and kernel space thread**
9. **Booting process in Linux**
10. **Memory management in Linux**
11. **Paging in Linux**
12. **Network File System calls in Linux**



1991
LINUS TORVALDS POSTS FAMOUS MESSAGE - *"HELLO EVERYBODY OUT THERE..."* - AND RELEASES FIRST LINUX CODE

1992
LINUS LICENSES LINUX UNDER THE GPL, AN IMPORTANT DECISION THAT WILL CONTRIBUTE TO ITS SUCCESS IN THE COMING YEARS

1993
SLACKWARE BECOMES FIRST WIDELY ADOPTED DISTRIBUTION

1996
LINUS VISITS AQUARIUM, GETS BIT BY A PENGUIN AND CHOOSES IT AS LINUX MASCOT

1998
TECH GIANTS BEGIN ANNOUNCING PLATFORM SUPPORT FOR LINUX

1999
RED HAT GOES PUBLIC

2003
IBM RUNS FAMOUS LINUX AD DURING THE SUPERBOWL

2005
LINUS APPEARS ON THE COVER OF BUSINESSWEEK WITH A STORY THAT HAILS LINUX AS A BUSINESS SUCCESS

2007
THE LINUX FOUNDATION IS FORMED TO PROMOTE PROTECT AND STANDARDIZE LINUX LINUS IS A FELLOW

2010
THE LINUX-BASED ANDROID OS OUTSHIPS ALL OTHER SMARTPHONE OSES IN THE U.S. AND CLIMBS TO DOMINANCE

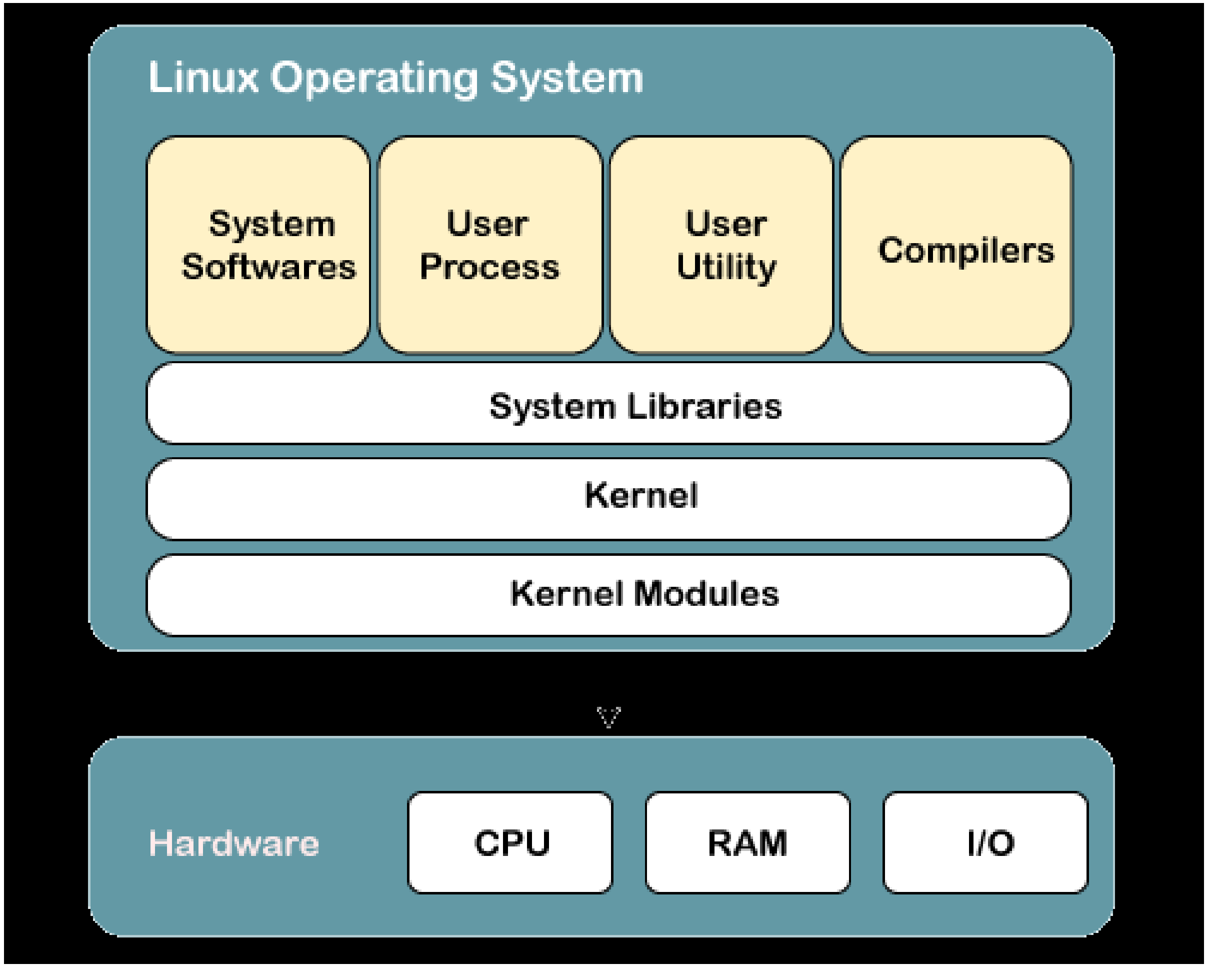
2011
LINUX TURNS 20 AND POWERS THE WORLD'S SUPERCOMPUTERS, STOCK EXCHANGES, PHONES, ATMS, HEALTHCARE RECORDS, SMART GRIDS, THE LIST GOES ON

20 YEARS OF LINUX
1991 - 2011

THE LINUX FOUNDATION
http://www.linuxfoundation.org/

Copyright © 2011 Linux Foundation. All rights reserved. Linux is a registered trademark of Linus Torvalds. Slackware® is a registered trademark of Slackware Linux, Inc. Red Hat® is a registered trademark of Red Hat, Inc. Android™ is a trademark of Google Inc. Use of this trademark is subject to Google Permissions. BusinessWeek is a trademark of BLOOMBERG L.P.

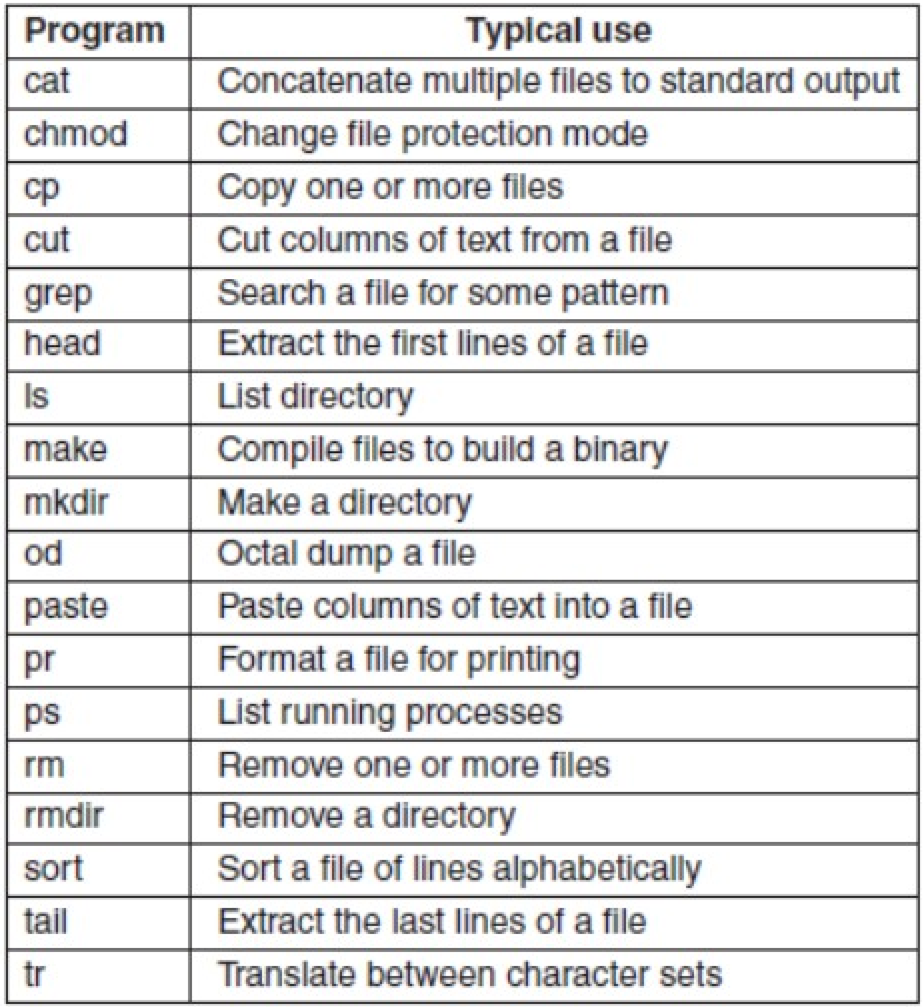
Linux architecture


Linux Operating System
System Softwares
User Process
User Utility
Compilers
System Libraries
Kernel
Kernel Modules
Hardware
CPU
RAM
I/O

Design goals of Linux

- Designed to handle multiple processes and multiple users at the same time
- Multi-user, multitasking system with a full set of UNIX-compatible tools.
- Speed, efficiency, and standardization.
- Designed to be compliant with the relevant POSIX documents.
- The Linux programming interface adheres to the SVR4 UNIX semantics.

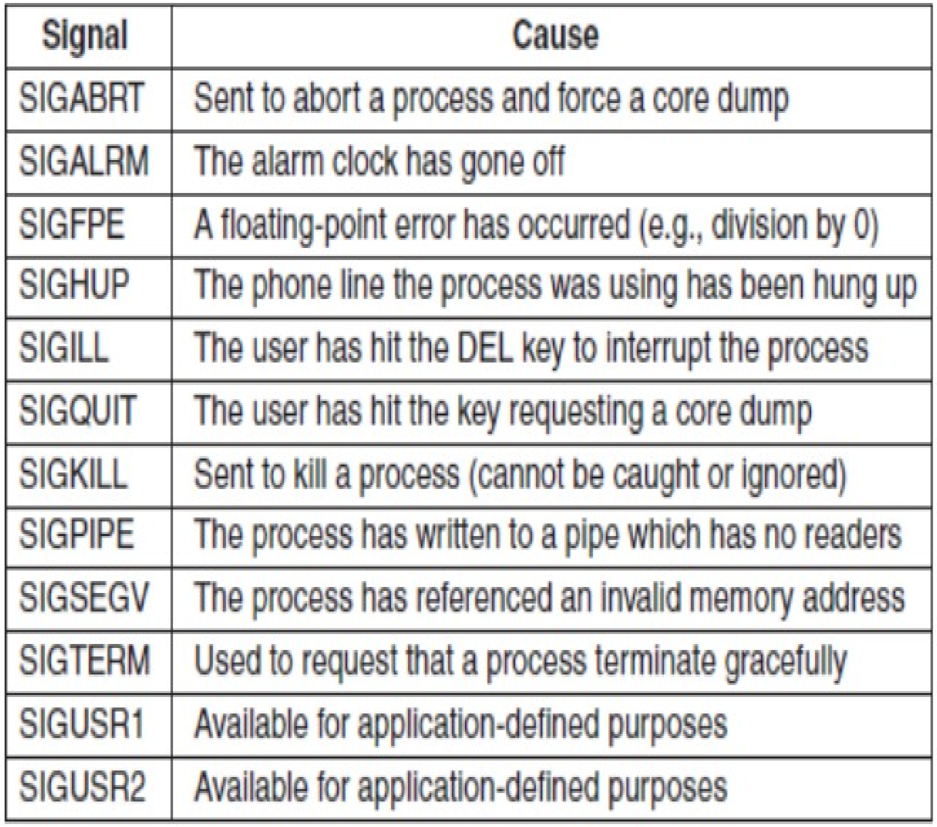
List of Linux Utility Programs



Program	Typical use
cat	Concatenate multiple files to standard output
chmod	Change file protection mode
cp	Copy one or more files
cut	Cut columns of text from a file
grep	Search a file for some pattern
head	Extract the first lines of a file
ls	List directory
make	Compile files to build a binary
mkdir	Make a directory
od	Octal dump a file
paste	Paste columns of text into a file
pr	Format a file for printing
ps	List running processes
rm	Remove one or more files
rmdir	Remove a directory
sort	Sort a file of lines alphabetically
tail	Extract the last lines of a file
tr	Translate between character sets

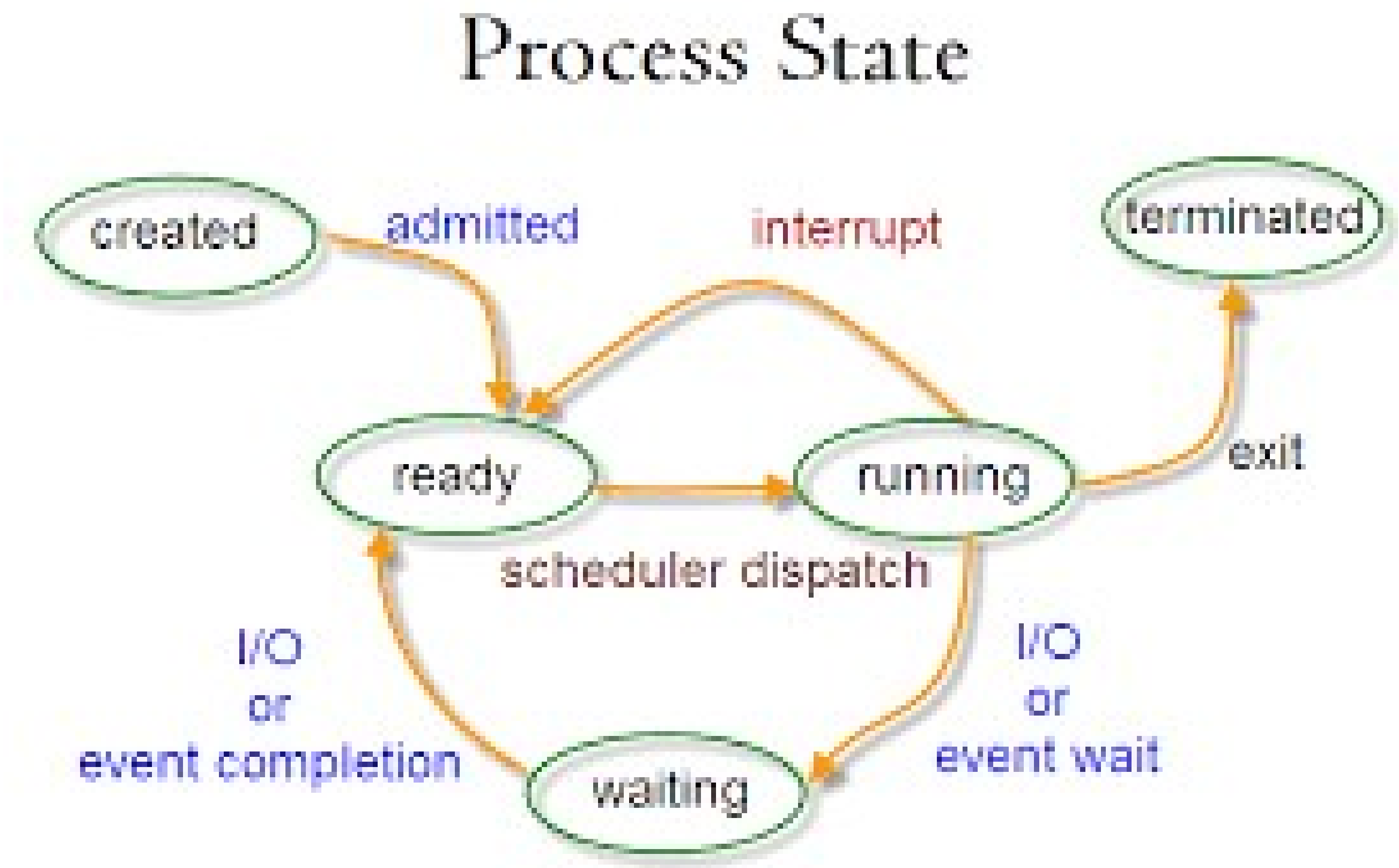
Process in Linux and PID, UID, GID in Linux

- The fork system call creates an exact copy of the original process.
- The forking process is called the parent process. The new process is called the child process.
- Process Identifier is when each process has a unique identifier associated with it known as process id.
- User and Group Identifiers (UID and GID) are the identifiers associated with a processes of the user and group.



Signal	Cause
SIGABRT	Sent to abort a process and force a core dump
SIGALRM	The alarm clock has gone off
SIGFPE	A floating-point error has occurred (e.g., division by 0)
SIGHUP	The phone line the process was using has been hung up
SIGILL	The user has hit the DEL key to interrupt the process
SIGQUIT	The user has hit the key requesting a core dump
SIGKILL	Sent to kill a process (cannot be caught or ignored)
SIGPIPE	The process has written to a pipe which has no readers
SIGSEGV	The process has referenced an invalid memory address
SIGTERM	Used to request that a process terminate gracefully
SIGUSR1	Available for application-defined purposes
SIGUSR2	Available for application-defined purposes

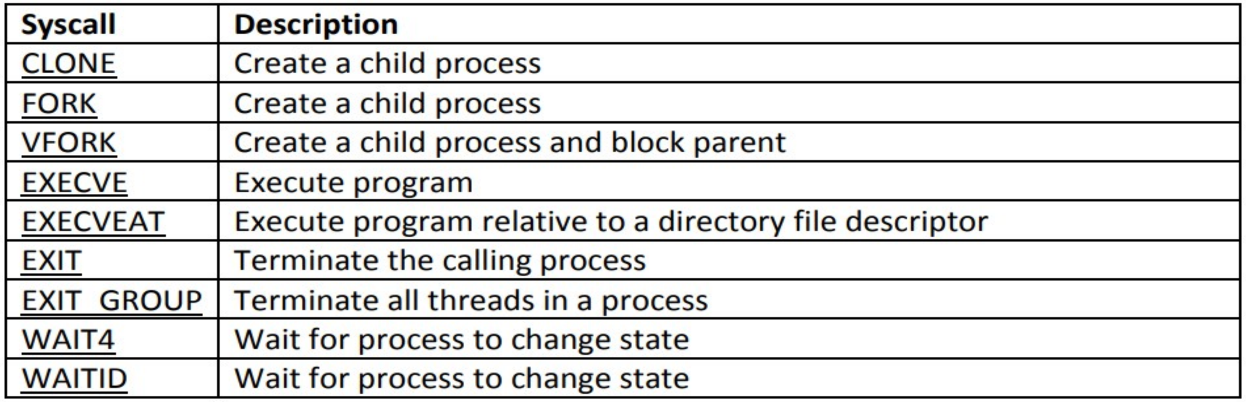
Linux Processes


Process State
created
admitted
interrupt
terminated
ready
running
exit
scheduler dispatch
I/O
or
event completion
I/O
or
event wait
waiting

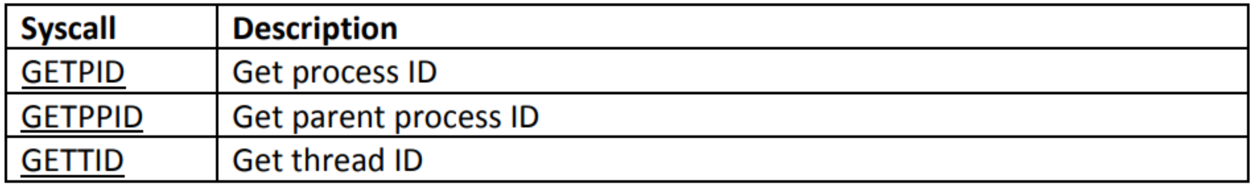
Linux Processes

- State
- Running
- Waiting
- Stopped
- Zombie – dead processes (not removed)
- Scheduling Information
- Identifiers
- Inter-Process Communication
- Links
- Times and Timers
- Virtual memory
- Processor Specific Context
- File system

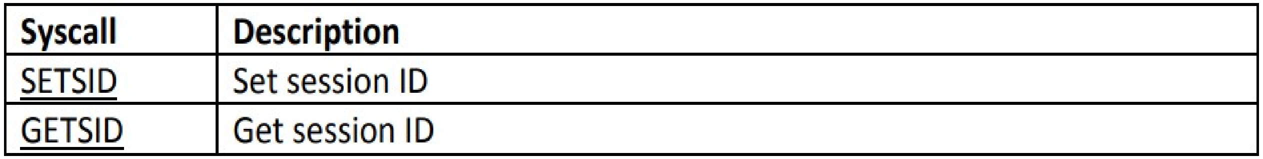
Process Management System Calls in Linux



Syscall	Description
CLONE	Create a child process
FORK	Create a child process
VFORK	Create a child process and block parent
EXECVE	Execute program
EXECVEAT	Execute program relative to a directory file descriptor
EXIT	Terminate the calling process
EXIT_GROUP	Terminate all threads in a process
WAIT4	Wait for process to change state
WAITID	Wait for process to change state

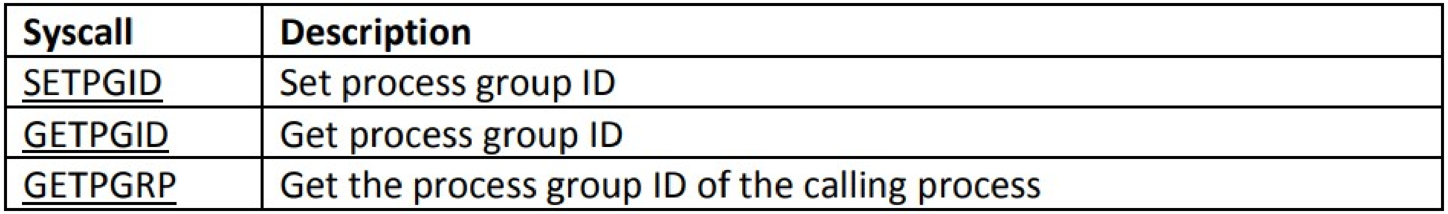


Syscall	Description
GETPID	Get process ID
GETPPID	Get parent process ID
GETTID	Get thread ID

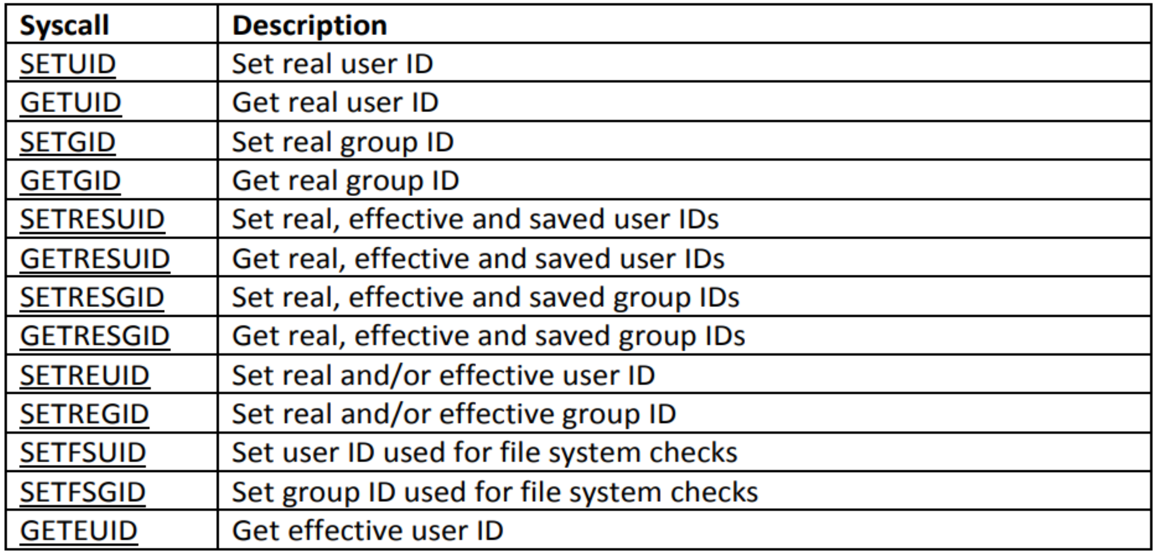


Syscall	Description
SETSID	Set session ID
GETSID	Get session ID

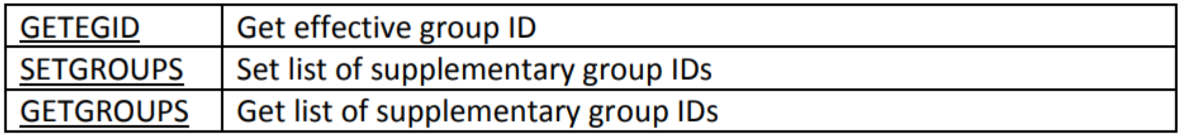
Process Management System Calls in Linux



Syscall	Description
SETPGID	Set process group ID
GETPGID	Get process group ID
GETPGRP	Get the process group ID of the calling process



Syscall	Description
SETUID	Set real user ID
GETUID	Get real user ID
SETGID	Set real group ID
GETGID	Get real group ID
SETRESUID	Set real, effective and saved user IDs
GETRESUID	Get real, effective and saved user IDs
SETRESGID	Set real, effective and saved group IDs
GETRESGID	Get real, effective and saved group IDs
SETREUID	Set real and/or effective user ID
SETREGID	Set real and/or effective group ID
SETFSUID	Set user ID used for file system checks
SETFSGID	Set group ID used for file system checks
GETEUID	Get effective user ID

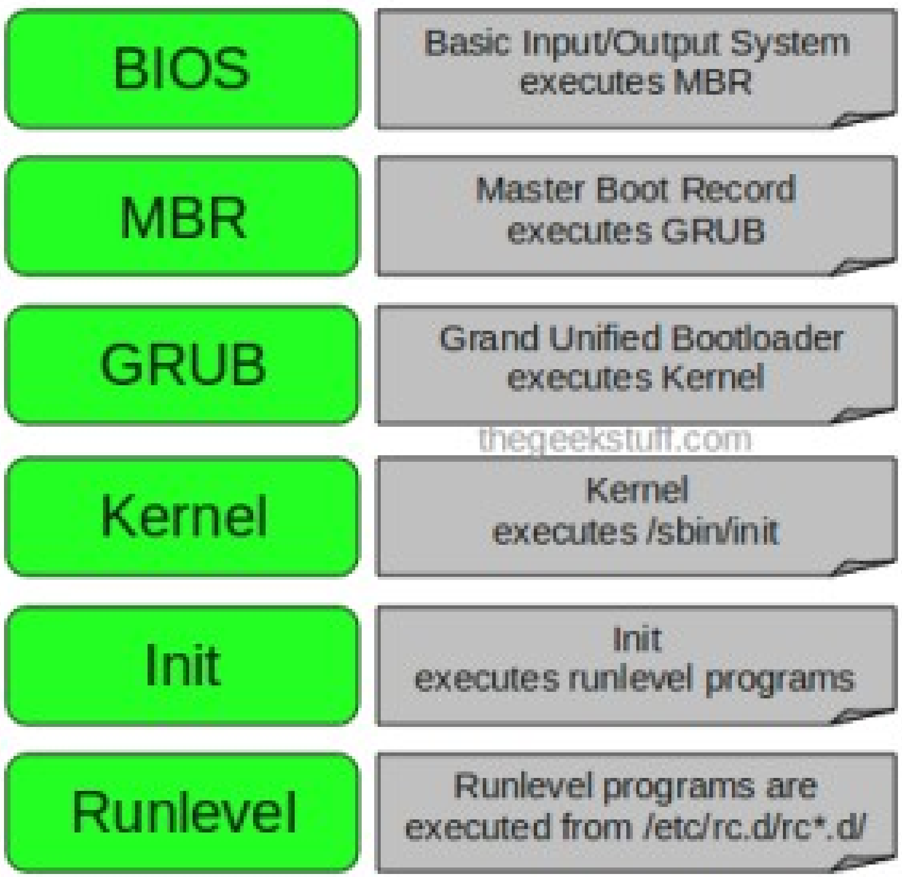


GETEGID	Get effective group ID
SETGROUPS	Set list of supplementary group IDs
GETGROUPS	Get list of supplementary group IDs

User and Kernel Space Thread

- **User Level Threads −** User managed threads.
- **Kernel Level Threads −** Operating System managed threads acting on kernel, an operating system core.
- **User-Space Threads:** cooperative multitasking, user threads typically can switch faster than kernel threads.
- **Kernel-space threads** often are implemented in the kernel using several tables.

Booting process in Linux

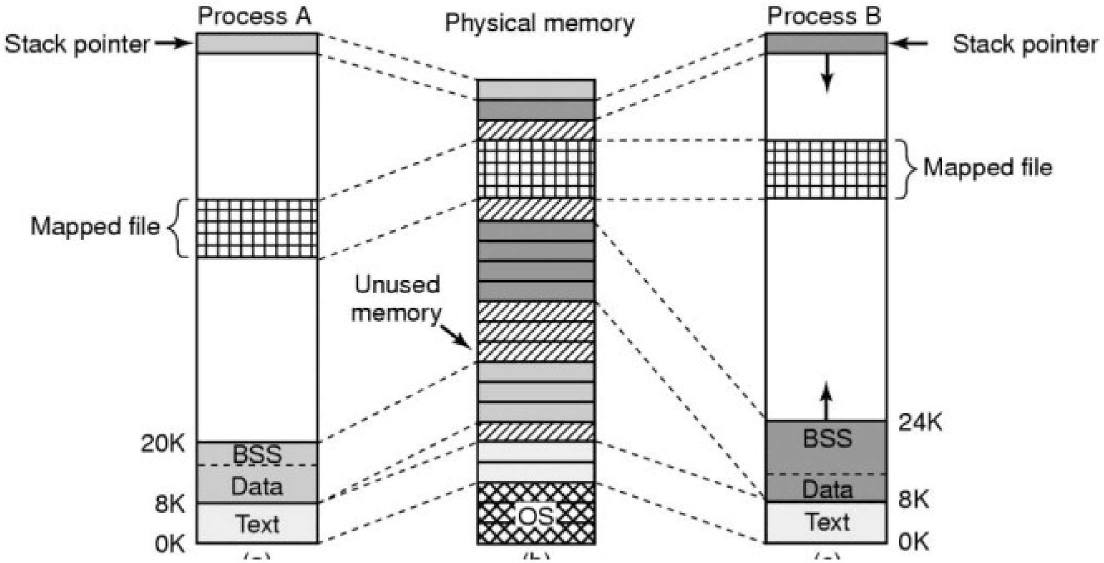


BIOS	Basic Input/Output System executes MBR
MBR	Master Boot Record executes GRUB
GRUB	Grand Unified Bootloader executes Kernel
Kernel	Kernel executes /sbin/init
Init	Init executes runlevel programs
Runlevel	Runlevel programs are executed from /etc/rc.d/rc*.d/

thegeekstuff.com

Memory Management in Linux

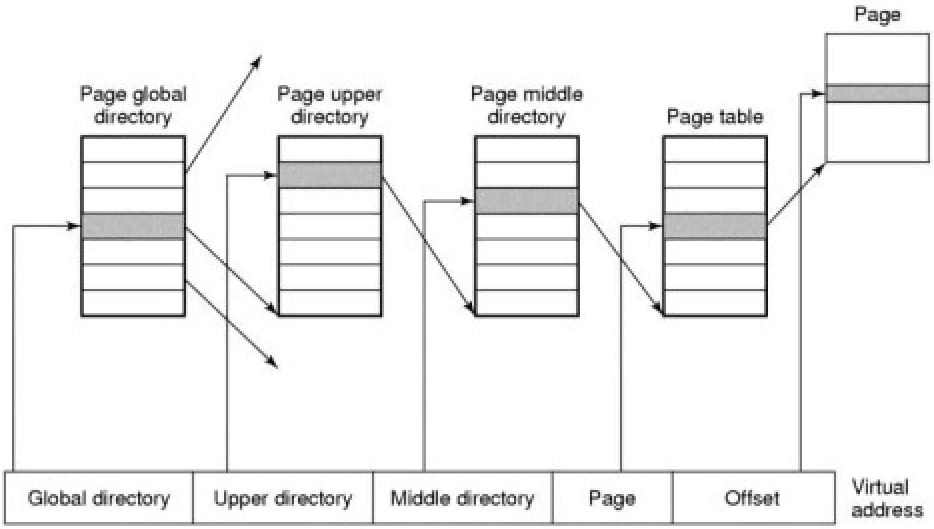
- The memory management subsystem is one of the most important parts of the operating system.
- Virtual memory makes the system appear to have more memory than it actually has by sharing it between competing processes as they need it.


Process A
Stack pointer
Mapped file
20K
BSS
Data
8K
Text
0K
Physical memory
Unused memory
OS
Process B
Stack pointer
Mapped file
24K
BSS
Data
8K
Text
0K

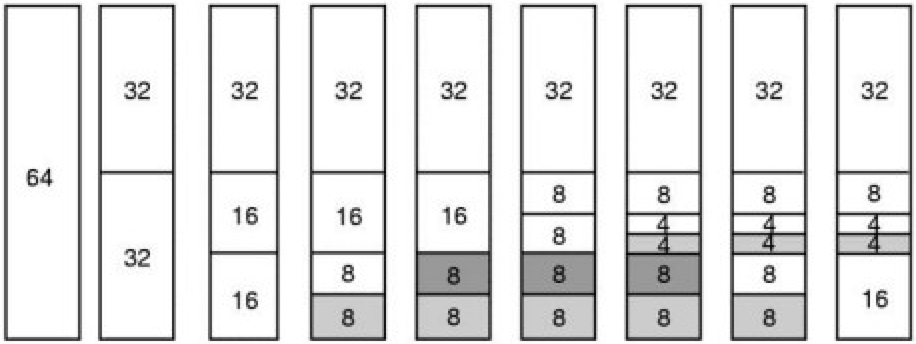
Paging in Linux

- The technique of only loading virtual pages into memory as they are accessed is known as demand paging.
- There are three kernel variables which control the paging operation: minfree, desfree, lotsfree.
- A demand paging system is quite similar to a paging system with swapping where processes reside in secondary memory and pages are loaded only on demand, not in advance.

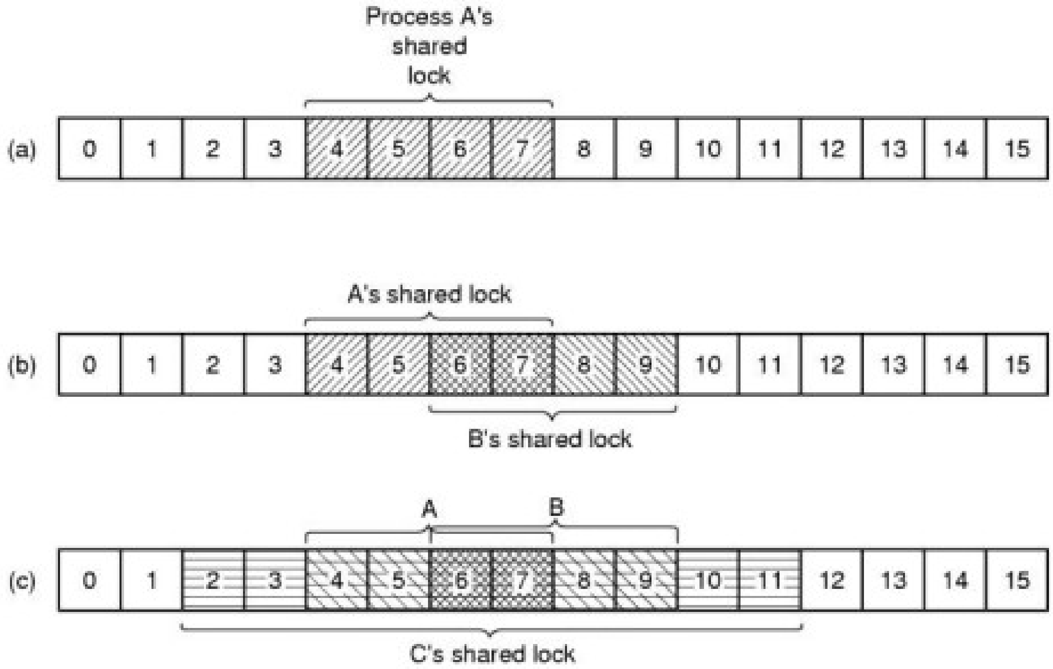
Physical Memory Management


Page
Page global directory
Page upper directory
Page middle directory
Page table
Global directory
Upper directory
Middle directory
Page
Offset
Virtual address

Memory Allocation Mechanisms


64
32
32
32
16
16
32
16
8
8
32
16
8
8
32
8
8
8
8
32
8
4
4
8
8
32
8
4
4
8
8
32
8
4
4
16

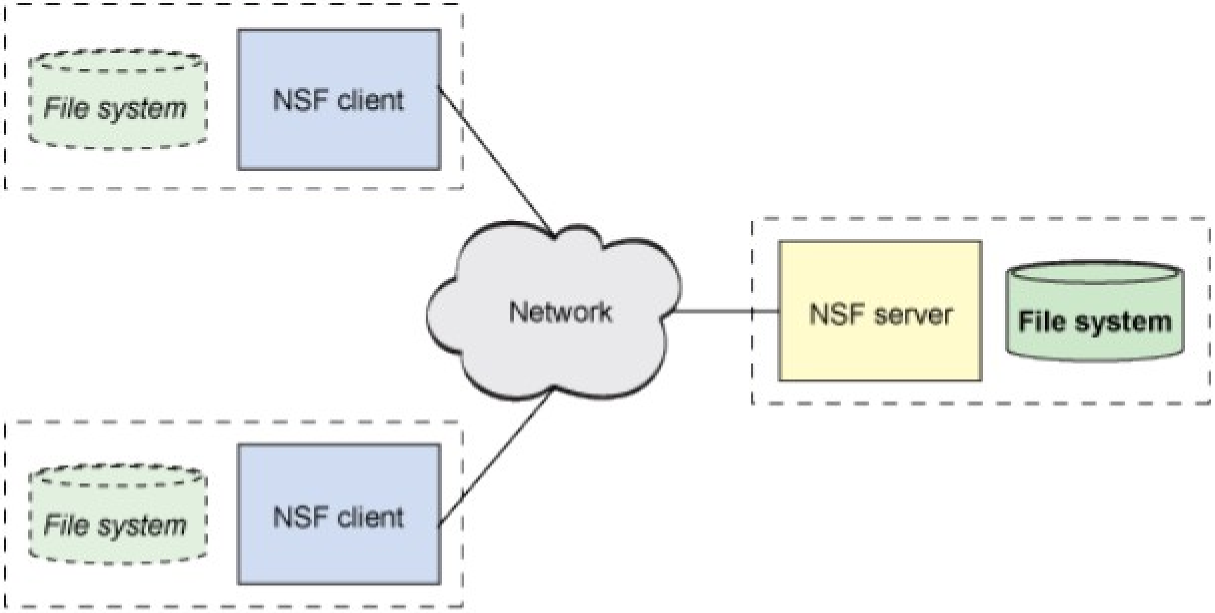
The Linux File System


Process A's shared lock
(a)
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
A's shared lock
(b)
B's shared lock
A
B
(c)
C's shared lock

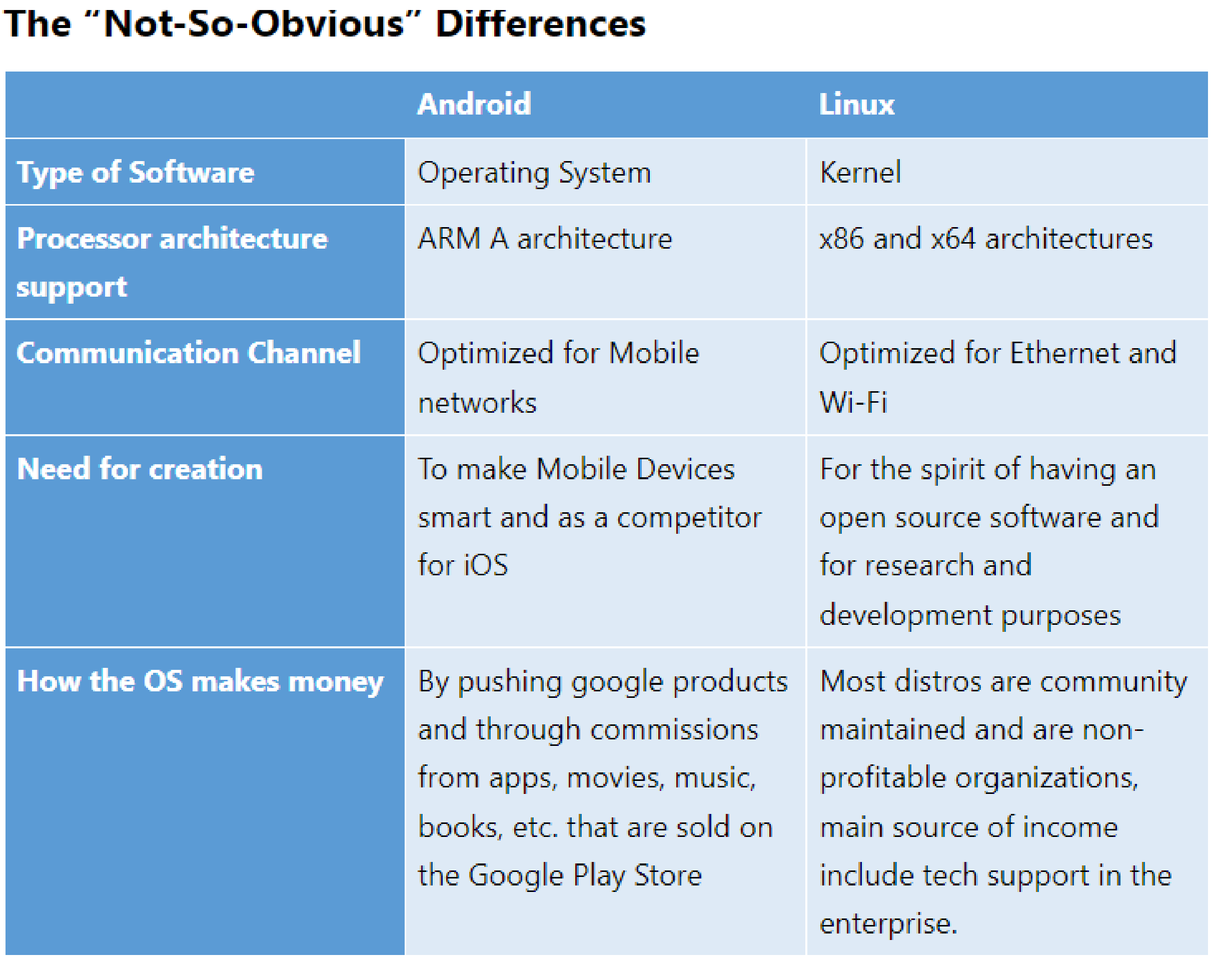
(a) A file with one lock. (b) Addition of a second lock. (c) A third lock.

Network File System (NFS) calls in Linux

- The Network File System (NFS) is a way of mounting Linux discs/directories over a network.
- An NFS server can export one or more directories that can then be mounted on a remote Linux machine.

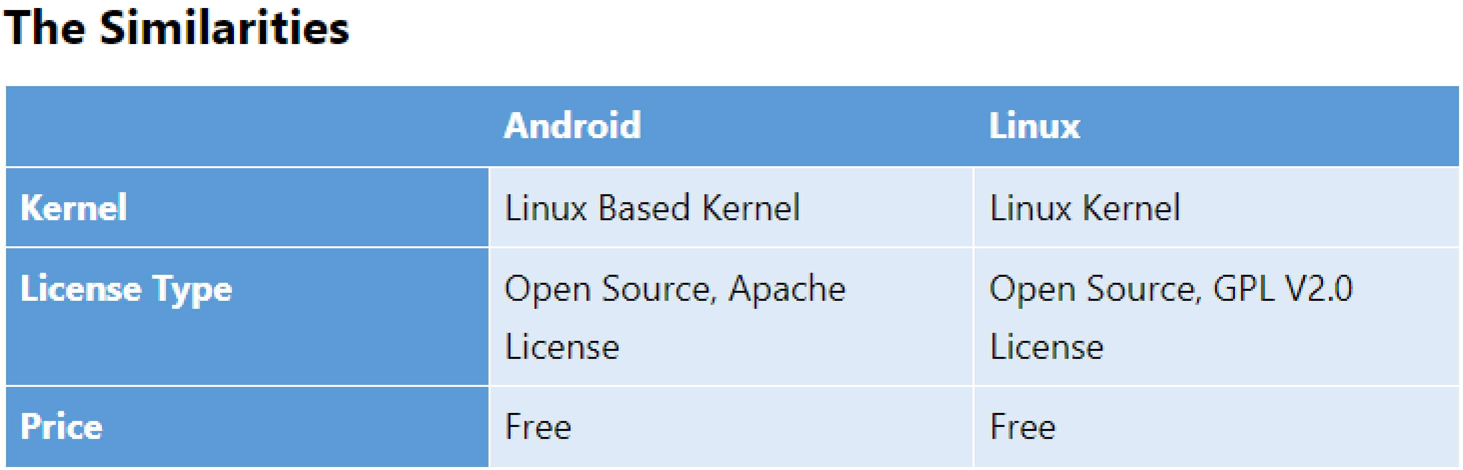

File system
NSF client
Network
NSF server
File system
File system
NSF client

Android vs Linux



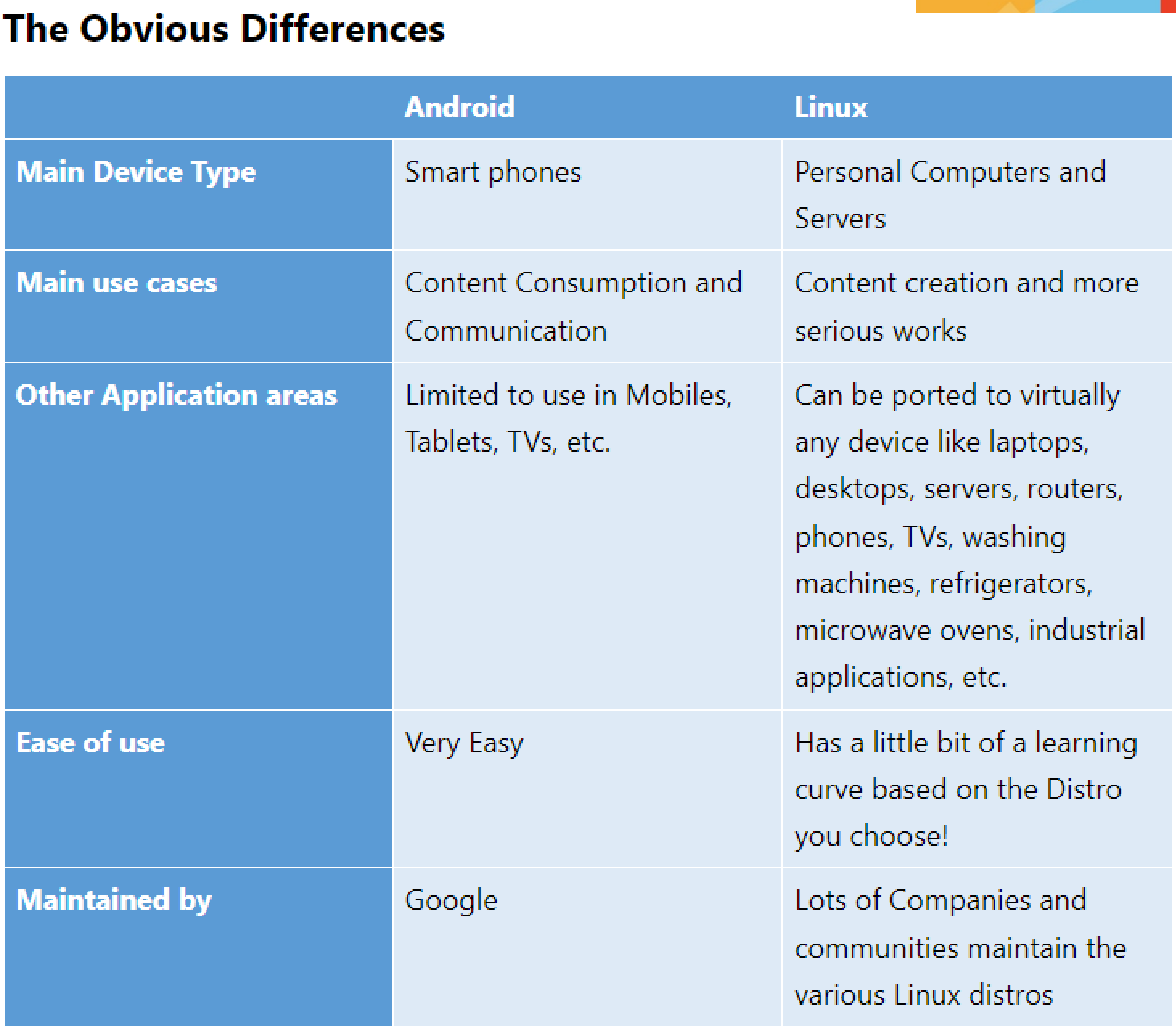
The “Not-So-Obvious” Differences

	Android	Linux
Type of Software	Operating System	Kernel
Processor architecture support	ARM A architecture	x86 and x64 architectures
Communication Channel	Optimized for Mobile networks	Optimized for Ethernet and Wi-Fi
Need for creation	To make Mobile Devices smart and as a competitor for iOS	For the spirit of having an open source software and for research and development purposes
How the OS makes money	By pushing google products and through commissions from apps, movies, music, books, etc. that are sold on the Google Play Store	Most distros are community maintained and are non-profitable organizations, main source of income include tech support in the enterprise.



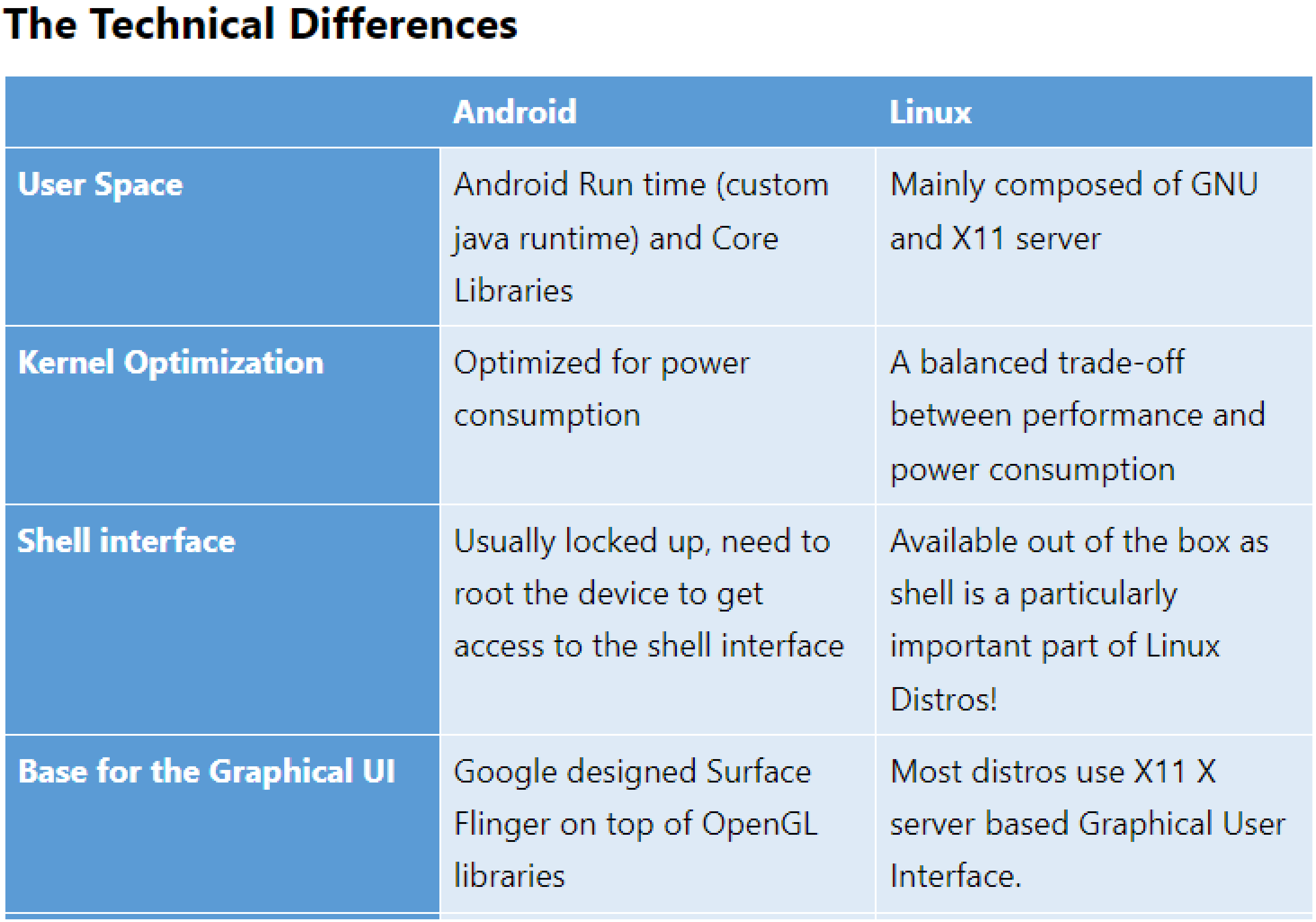
The Similarities

	Android	Linux
Kernel	Linux Based Kernel	Linux Kernel
License Type	Open Source, Apache License	Open Source, GPL V2.0 License
Price	Free	Free



The Obvious Differences

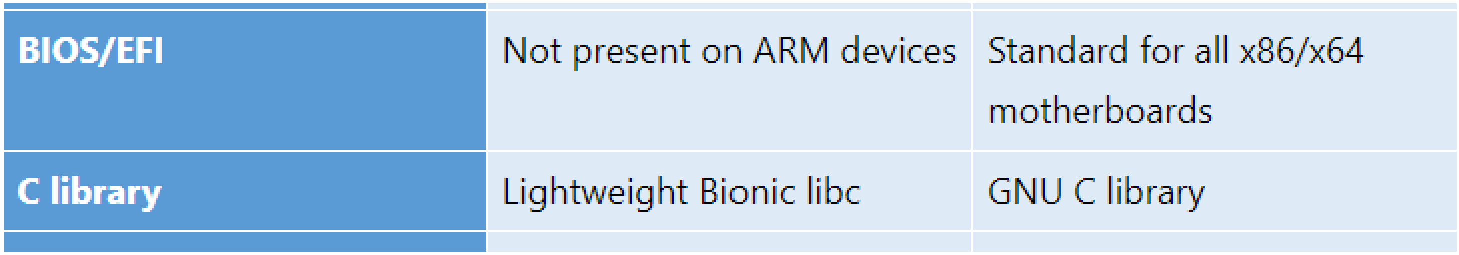
	Android	Linux
Main Device Type	Smart phones	Personal Computers and Servers
Main use cases	Content Consumption and Communication	Content creation and more serious works
Other Application areas	Limited to use in Mobiles, Tablets, TVs, etc.	Can be ported to virtually any device like laptops, desktops, servers, routers, phones, TVs, washing machines, refrigerators, microwave ovens, industrial applications, etc.
Ease of use	Very Easy	Has a little bit of a learning curve based on the Distro you choose!
Maintained by	Google	Lots of Companies and communities maintain the various Linux distros



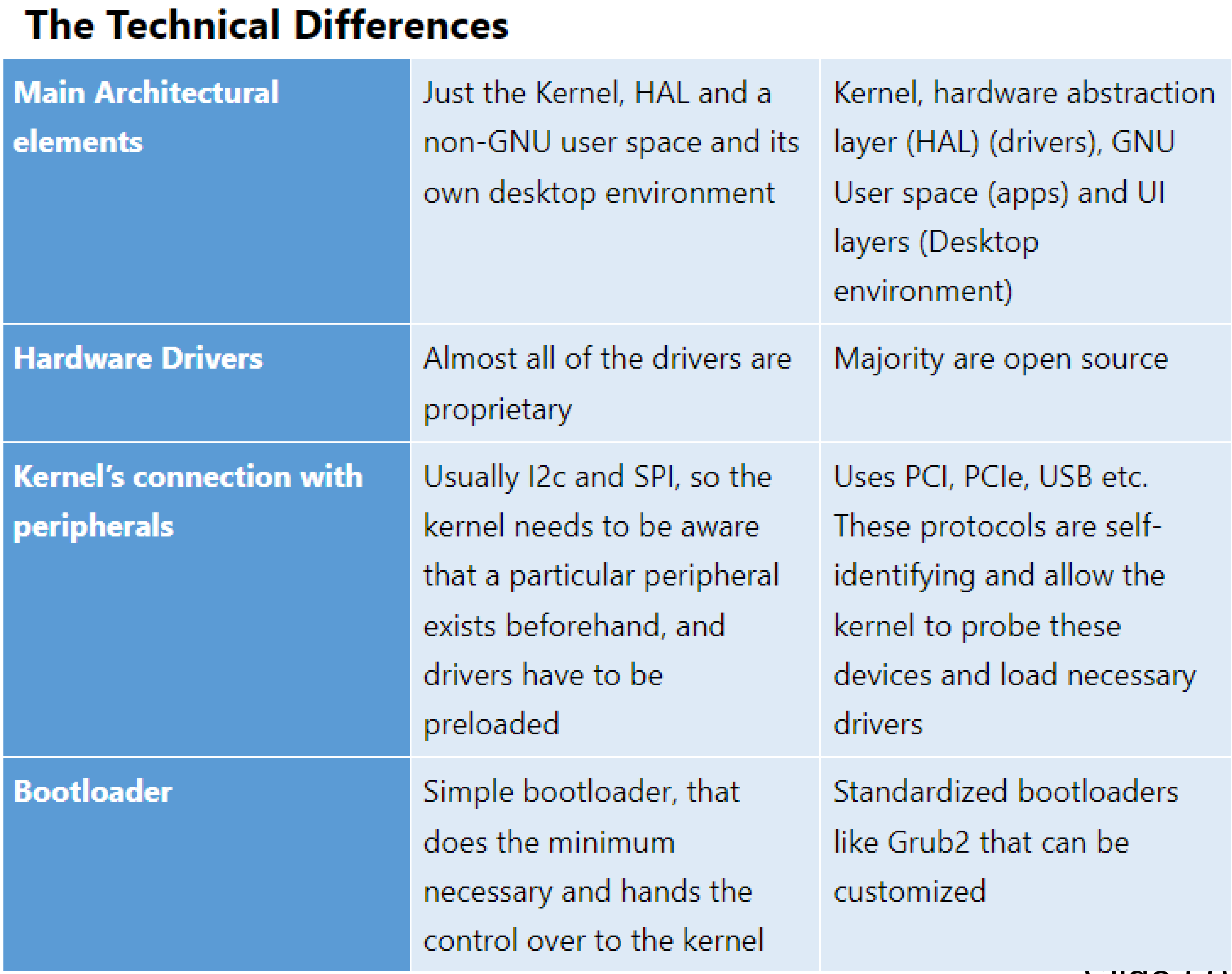
The Technical Differences

	Android	Linux
User Space	Android Run time (custom java runtime) and Core Libraries	Mainly composed of GNU and X11 server
Kernel Optimization	Optimized for power consumption	A balanced trade-off between performance and power consumption
Shell interface	Usually locked up, need to root the device to get access to the shell interface	Available out of the box as shell is a particularly important part of Linux Distros!
Base for the Graphical UI	Google designed Surface Flinger on top of OpenGL libraries	Most distros use X11 X server based Graphical User Interface.

The Technical Differences



BIOS/EFI	Not present on ARM devices	Standard for all x86/x64 motherboards
C library	Lightweight Bionic libc	GNU C library



The Technical Differences

Main Architectural elements	Just the Kernel, HAL and a non-GNU user space and its own desktop environment	Kernel, hardware abstraction layer (HAL) (drivers), GNU User space (apps) and UI layers (Desktop environment)
Hardware Drivers	Almost all of the drivers are proprietary	Majority are open source
Kernel's connection with peripherals	Usually I2c and SPI, so the kernel needs to be aware that a particular peripheral exists beforehand, and drivers have to be preloaded	Uses PCI, PCIe, USB etc. These protocols are self-identifying and allow the kernel to probe these devices and load necessary drivers
Bootloader	Simple bootloader, that does the minimum necessary and hands the control over to the kernel	Standardized bootloaders like Grub2 that can be customized

THANK YOU

BITS Pilani
Pilani Campus

COMPUTING AND DESIGN

SESSION 7

Course design by - Dr. Lucy J. Gudino &
Dr Chandrasekhar
Faculty – Balamurali Shankar
WILP & Department of CS & IS

BITS Pilani
Pilani Campus

BITS Pilani

Pilani Campus

Operating systems – Memory Interface

Last Session

List of Topic Title
Hardware Abstractions (Contd…) Case Study 1: Linux on Intel processor Case Study 2: Android on a mobile chipset

Memory Interface

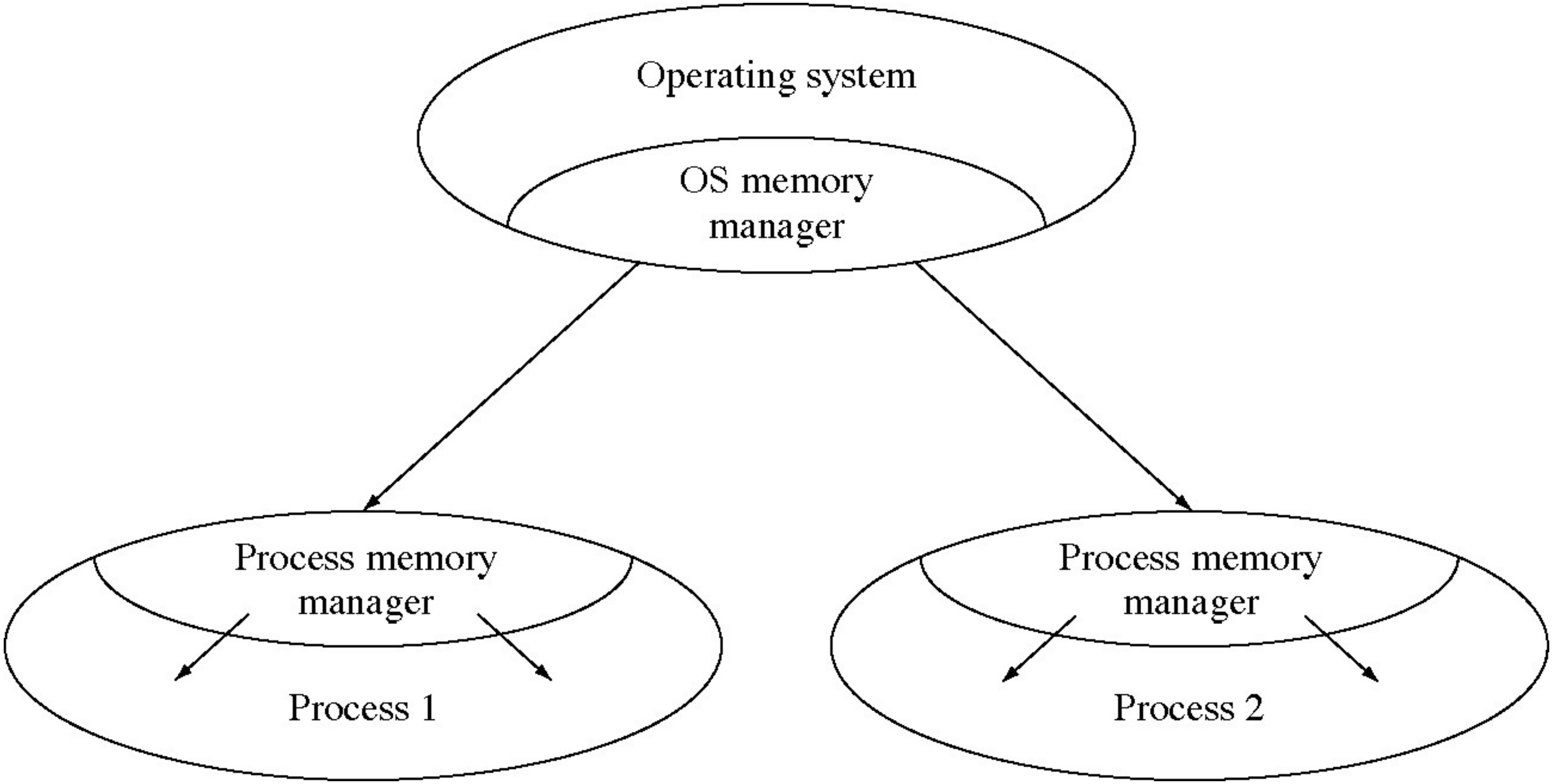
Today’s topic

- Levels of memory management
- The memory management design problem
- Solution to the memory management design problem
- Dynamic memory allocation

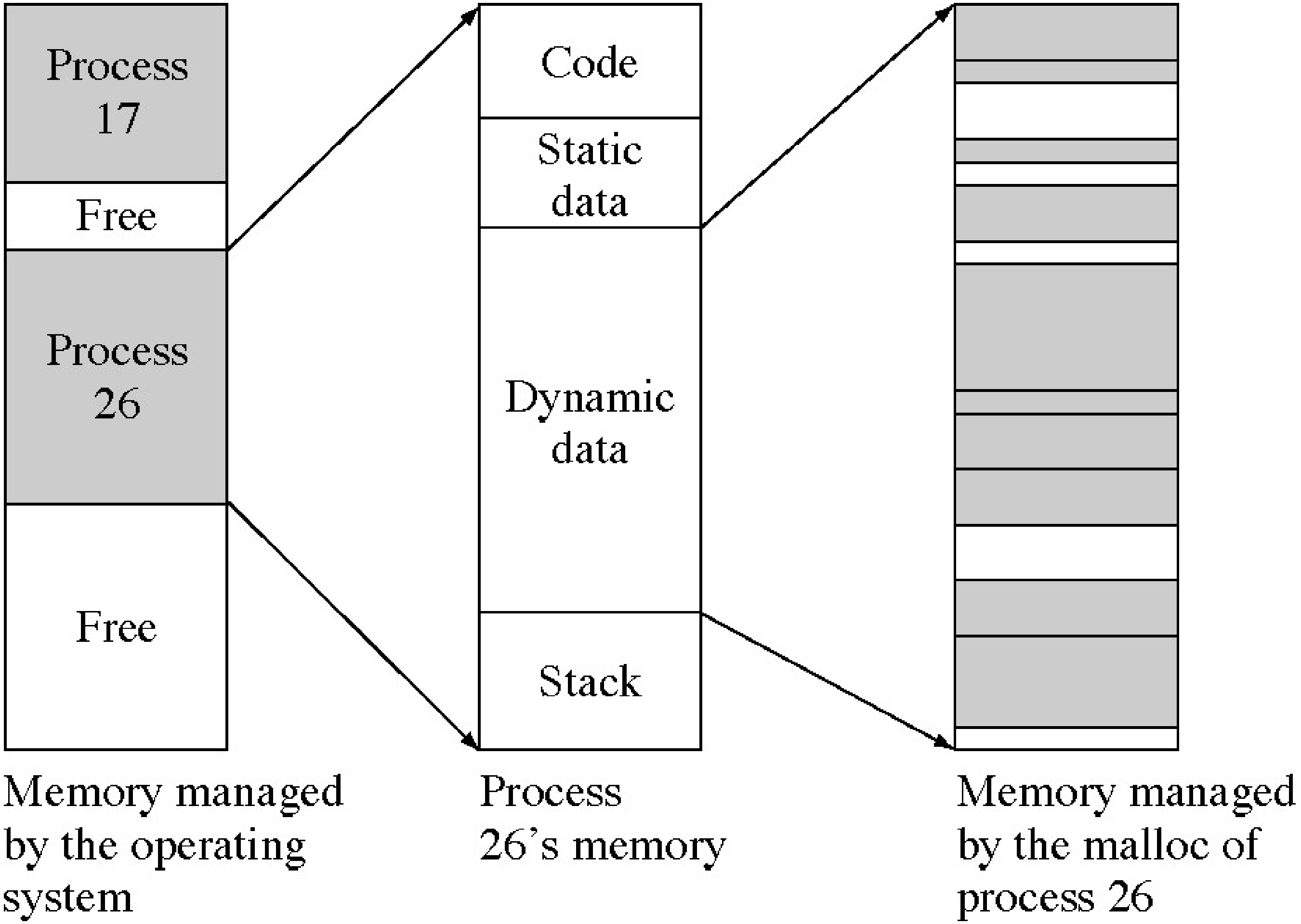
Levels of Memory management

BITS Pilani
Pilani Campus

Two levels of memory management


Operating system
OS memory manager
Process memory manager
Process 1
Process memory manager
Process 2

Two levels of memory management

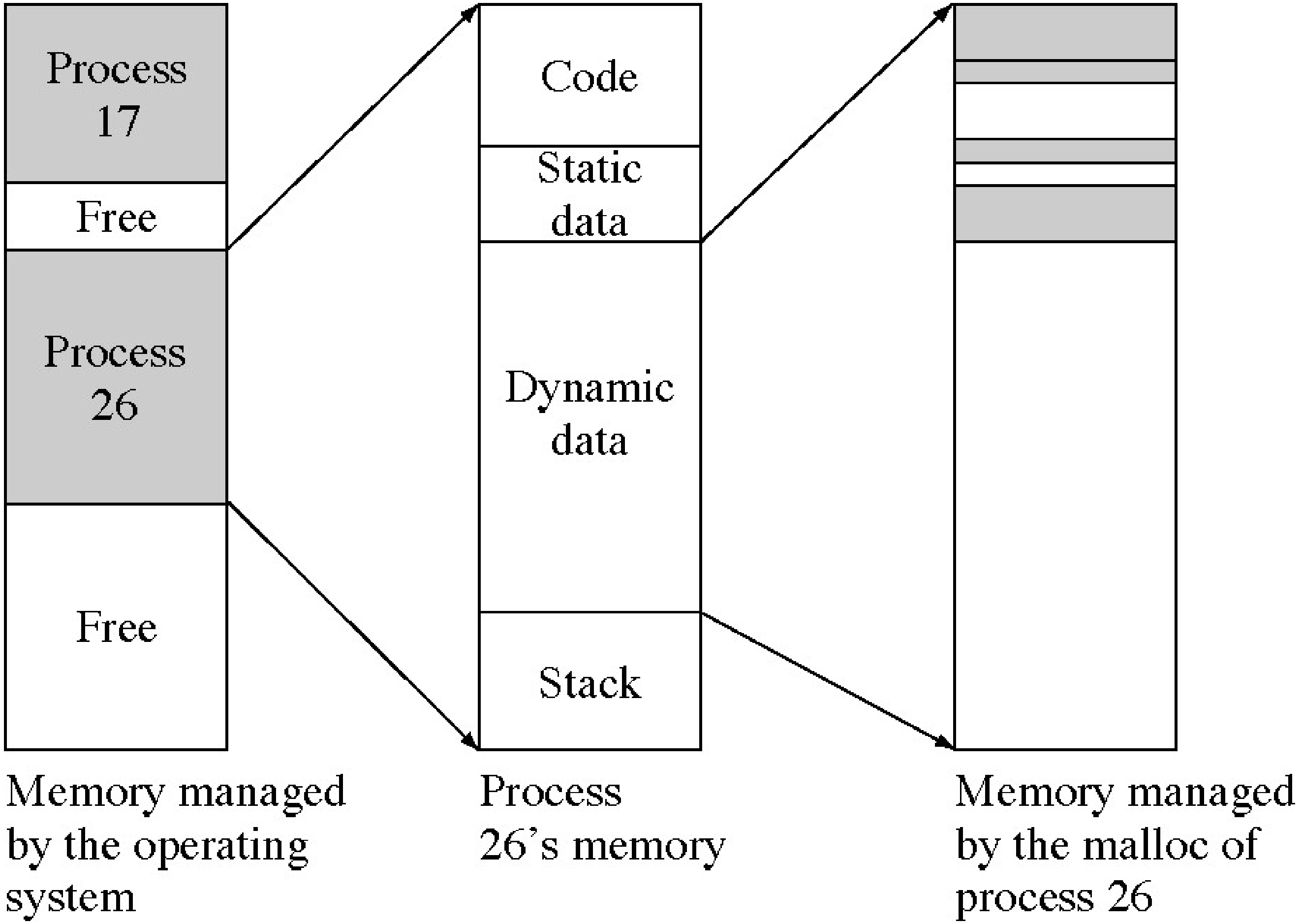

Process 17
Free
Process 26
Free
Code
Static data
Dynamic data
Stack

Memory managed by the operating system

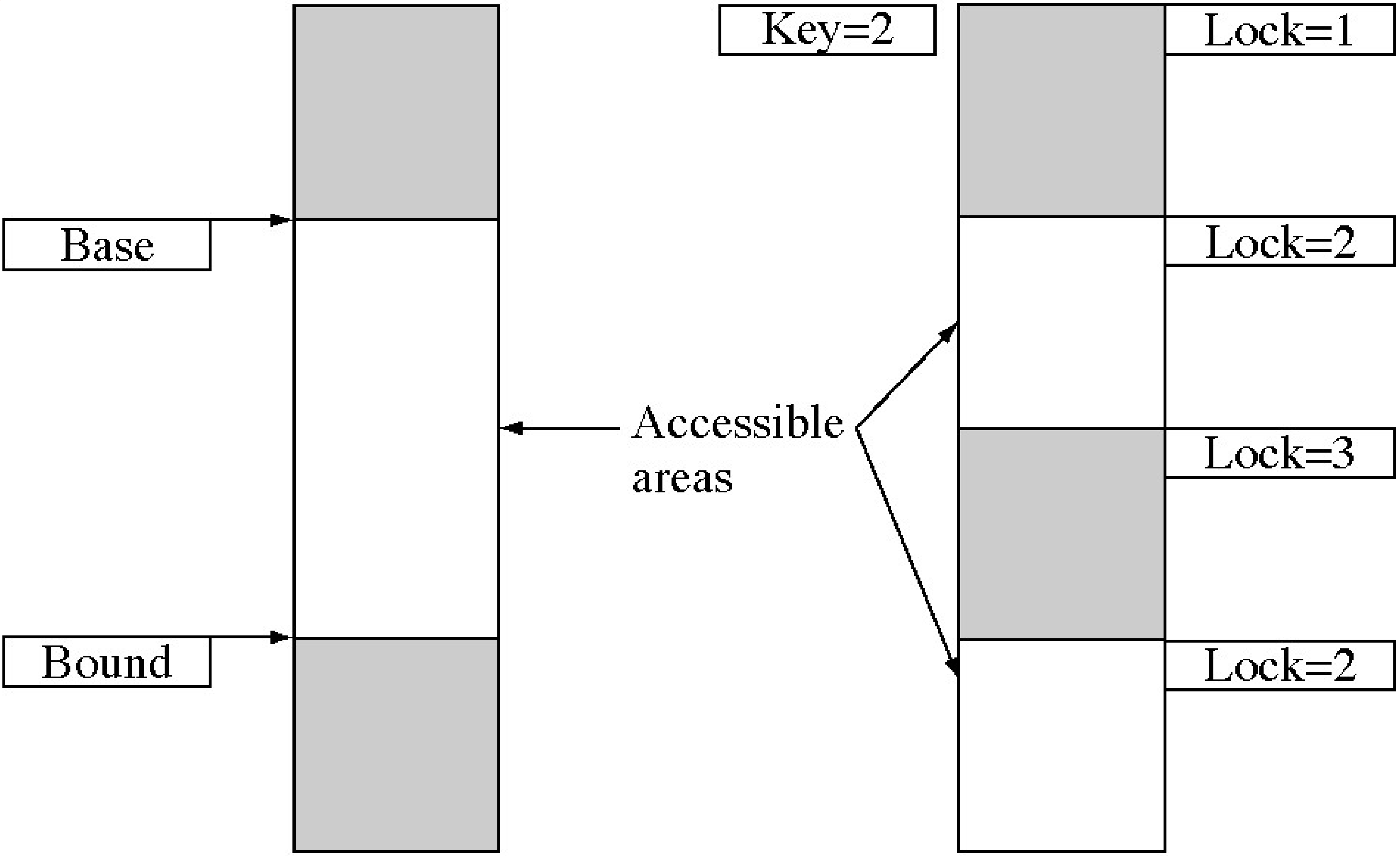
Process 26's memory

Memory managed by the malloc of process 26

Free memory at the malloc level, but not at the OS level


Process
17
Free
Process
26
Free
Code
Static
data
Dynamic
data
Stack
Memory managed by the operating system
Process 26's memory
Memory managed by the malloc of process 26

Two forms of memory protection


Key=2
Lock=1
Base
Lock=2
Accessible
areas
Lock=3
Bound
Lock=2

(a) Base and bound registers **(b)** Lock and key registers

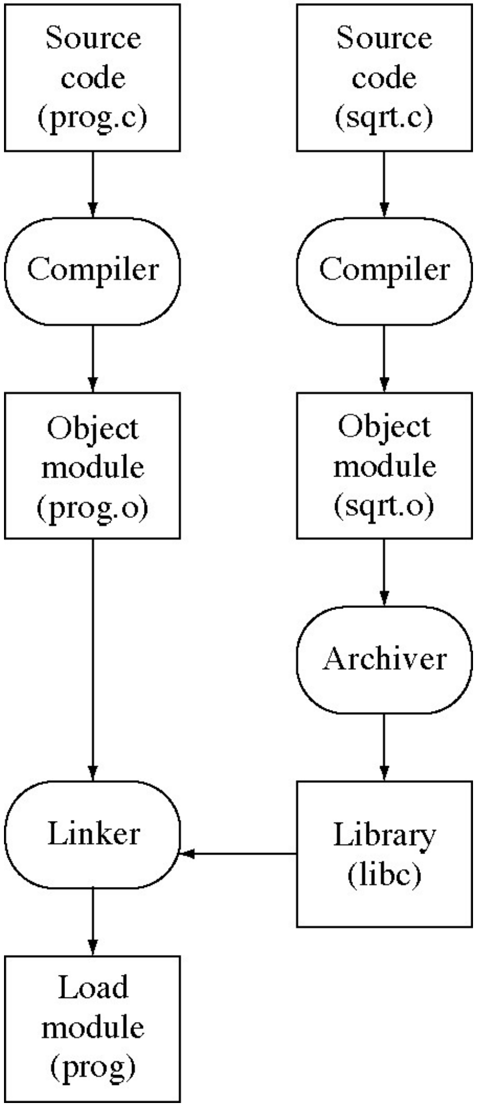
Memory management design problem

BITS Pilani
Pilani Campus

Memory Management Requirements

- Five requirements
    - Relocation
    - Protection
    - Sharing
    - Logical organization
    - Physical organization

Creating a load module


Source code (prog.c)
Source code (sqrt.c)
Compiler
Compiler
Object module (prog.o)
Object module (sqrt.o)
Archiver
Linker
Library (libc)
Load module (prog)

Object module format

- Header information
- Machine code
- Initialized data
- Symbol table
- Relocation information

Sample C++ program

-
  ```
  #include <iostream.h>
  #include <math.h>
  float arr[100];
  int size = 100;
  void main(int argc,char **argv) {
    int i; float sum = 0;
    for(I=0; I<size; ++i) {
      cin >> arr[i];//or “>>”(cin,arr[i])
      arr[i] = sqrt(arr[i]);
      sum += arr[i];
    }
    cout << sum;// or “<<“(cout,sum)
  }
  ```

Linker function

- Combine the object modules into a load module
- Relocate the object modules as they are being loaded
- Link the object modules together as they are being loaded
- Search libraries for external references not defined in the object modules

Linker algorithm (1 of 3)

- 1. Initialize by creating an empty load module and an empty symbol table
- 2. Read the next object module or library name from the command line

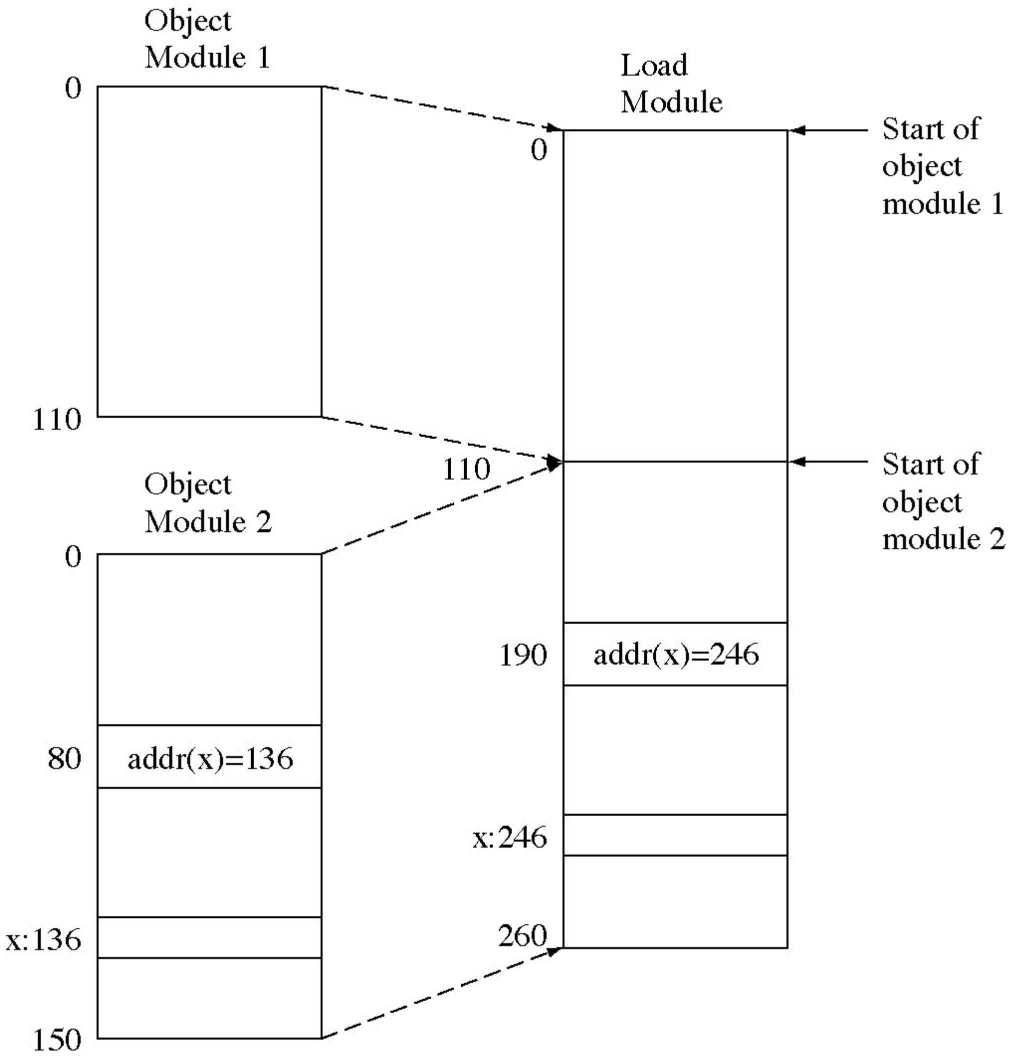
Linker algorithm (2 of 3)

- 3. If it is an object module then:
  - a. Insert it into the load module
  - b. Relocate it and its symbols
  - c. merge its symbol table into the global symbol table
  - d. For each undefined external reference in the object module's symbol table:
    - (1) If the symbol is already in the global symbol table then copy the value to the object module.
    - (2) If not then insert it (as undefined) into the global symbol table and make a note to fix up the symbol late
  - e. For each defined symbol in the object module, fix up all previous references to the symbol (in object modules loaded earlier).

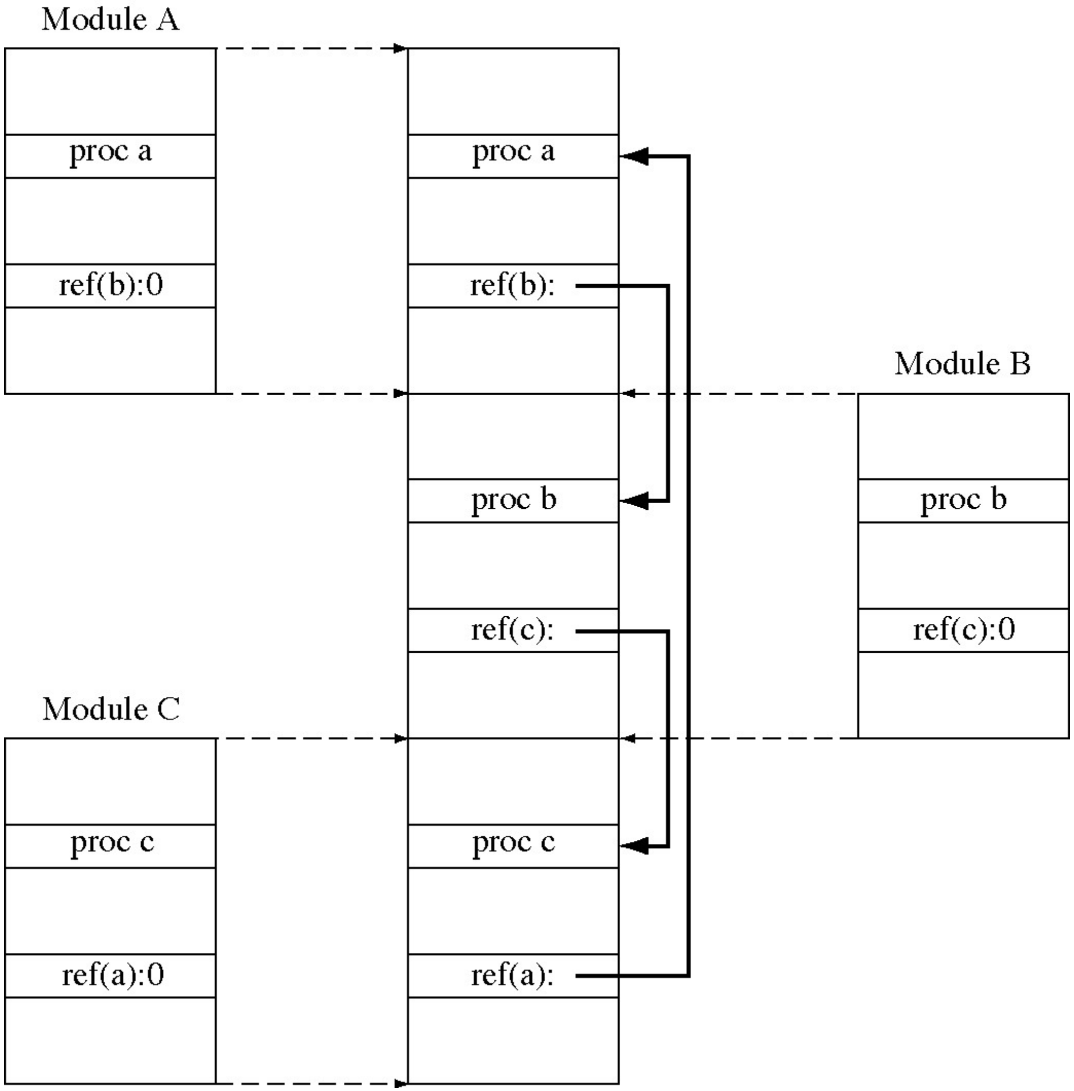
Linker algorithm (3 of 3)

- 4. If it is a library then:
  - a. Find each undefined symbol in the global symbol table
  - b. See if the symbol is defined in a module in this library
  - c. If so, the load the object module as described in step 3.
- 5. Go back to step 2.

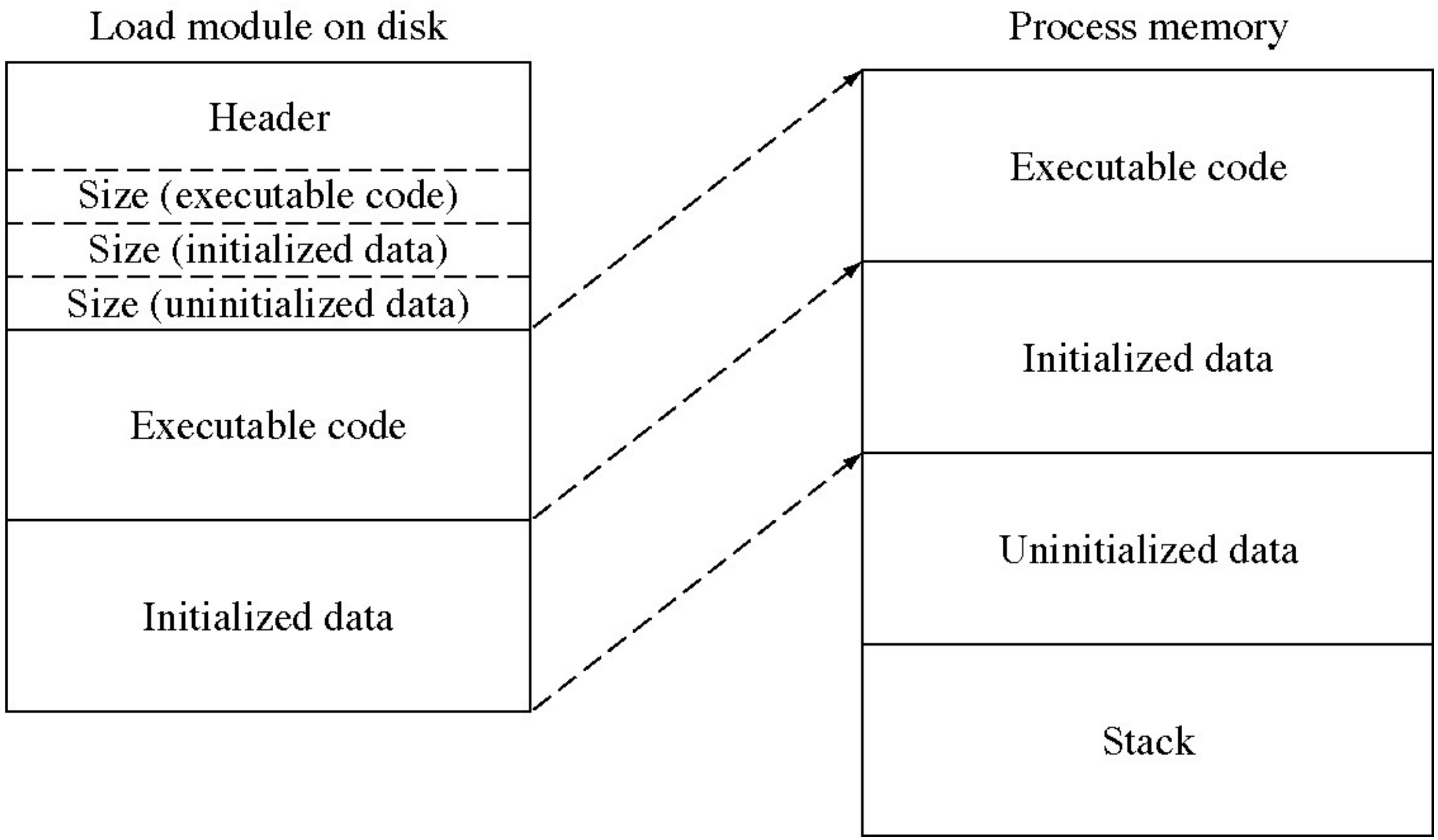
Relocation


Object
Module 1
0
110
Object
Module 2
0
80
addr(x)=136
x:136
150
Load
Module
0
110
190
addr(x)=246
x:246
260
Start of
object
module 1
Start of
object
module 2

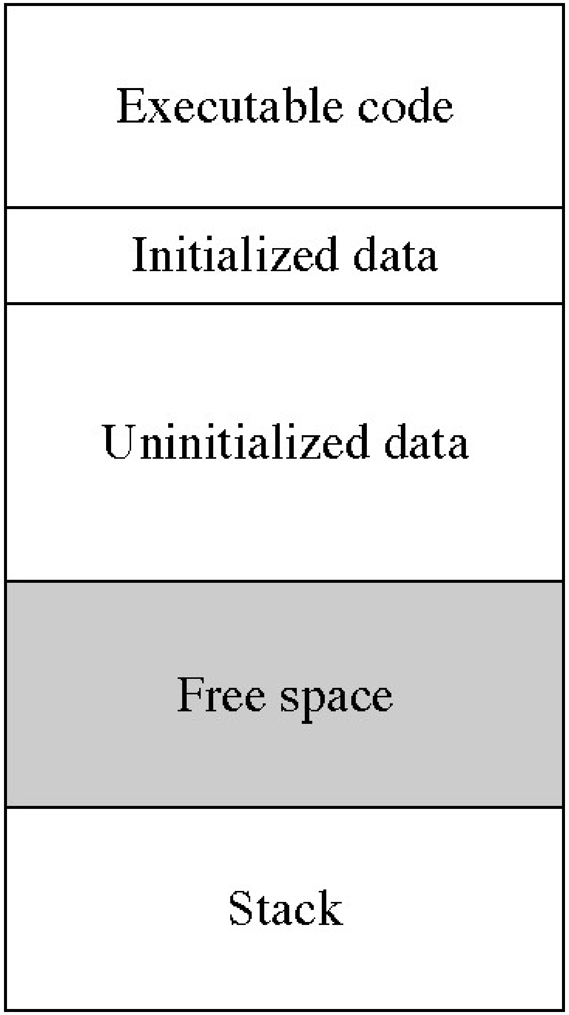
Linking


Module A
proc a
ref(b):0
proc a
ref(b):
Module B
proc b
proc b
ref(c):
ref(c):0
Module C
proc c
proc c
ref(a):0
ref(a):

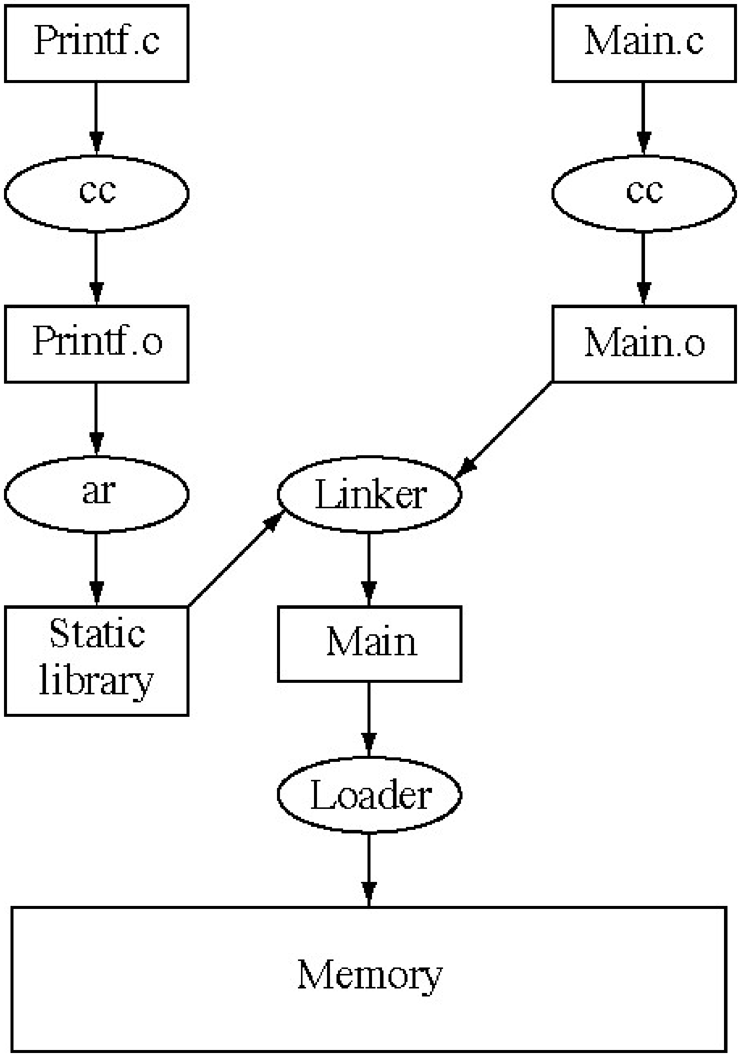
Loading a program into memory


Load module on disk
Header
Size (executable code)
Size (initialized data)
Size (uninitialized data)
Executable code
Initialized data
Process memory
Executable code
Initialized data
Uninitialized data
Stack

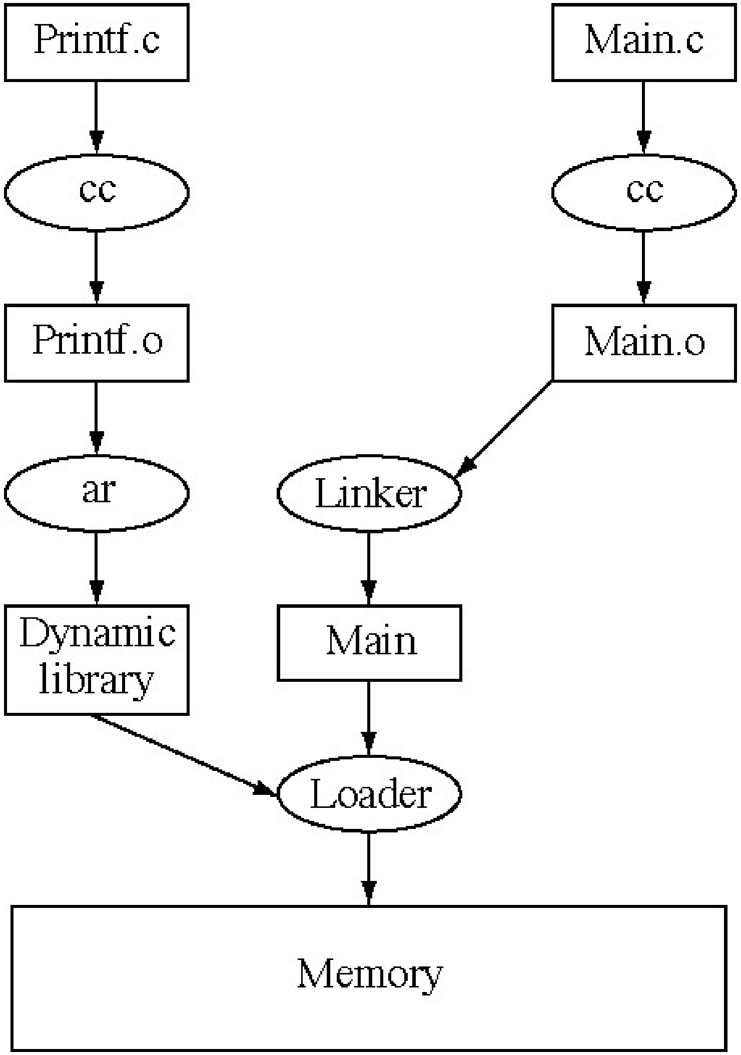
Memory areas in a running process


Executable code
Initialized data
Uninitialized data
Free space
Stack

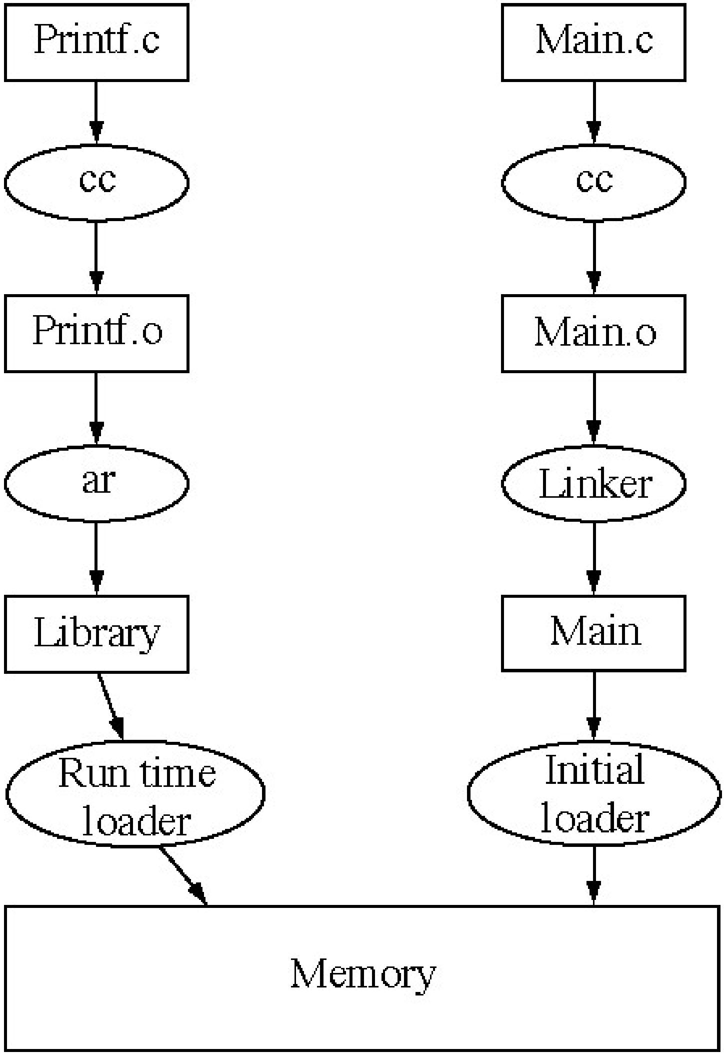
Normal linking and loading


Printf.c
cc
Printf.o
ar
Static library
Main.c
cc
Main.o
Linker
Main
Loader
Memory

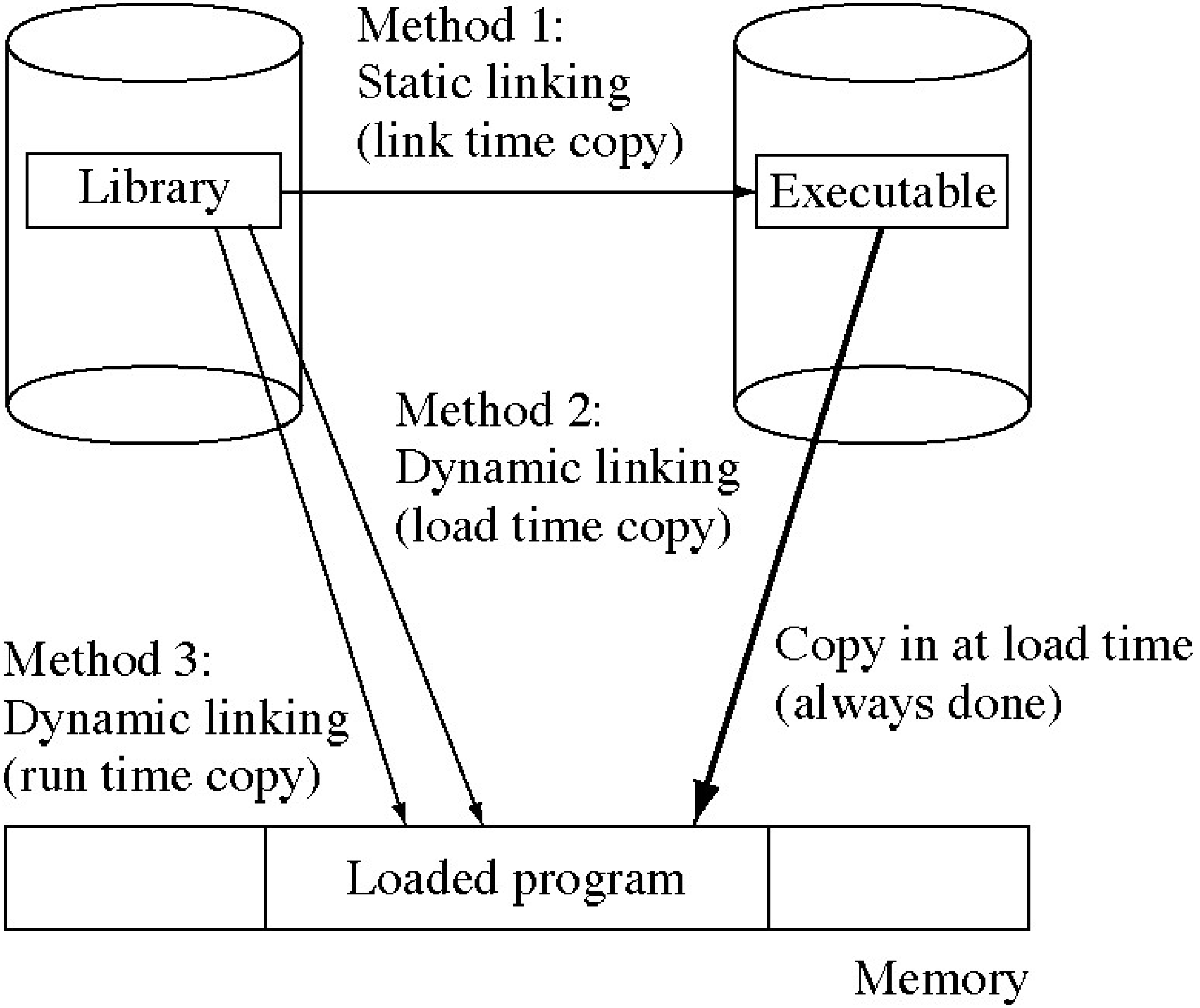
Load-time dynamic linking


Printf.c
cc
Printf.o
ar
Dynamic library
Main.c
cc
Main.o
Linker
Main
Loader
Memory

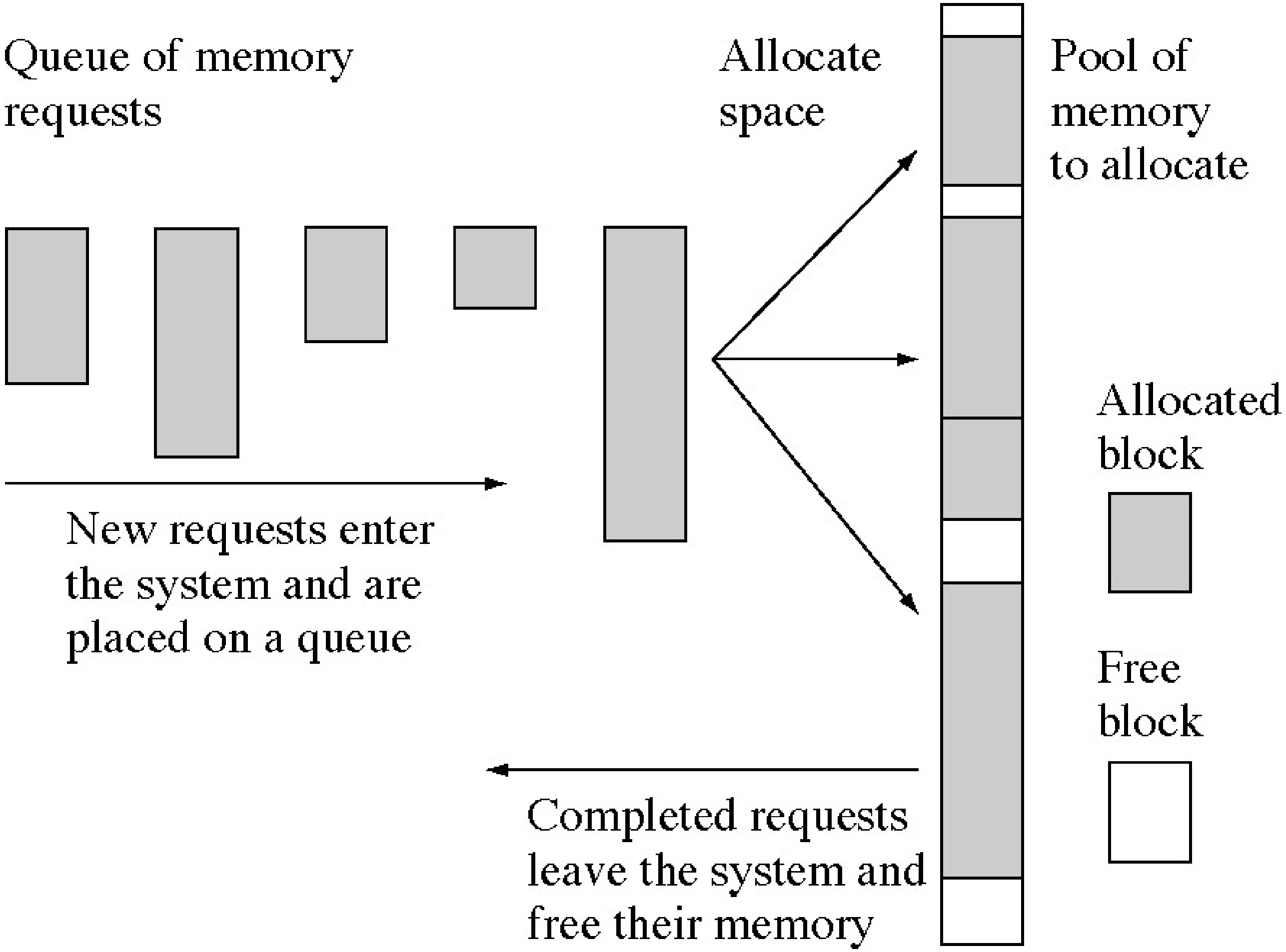
Run-time dynamic linking


Printf.c
Main.c
cc
cc
Printf.o
Main.o
ar
Linker
Library
Main
Run time loader
Initial loader
Memory

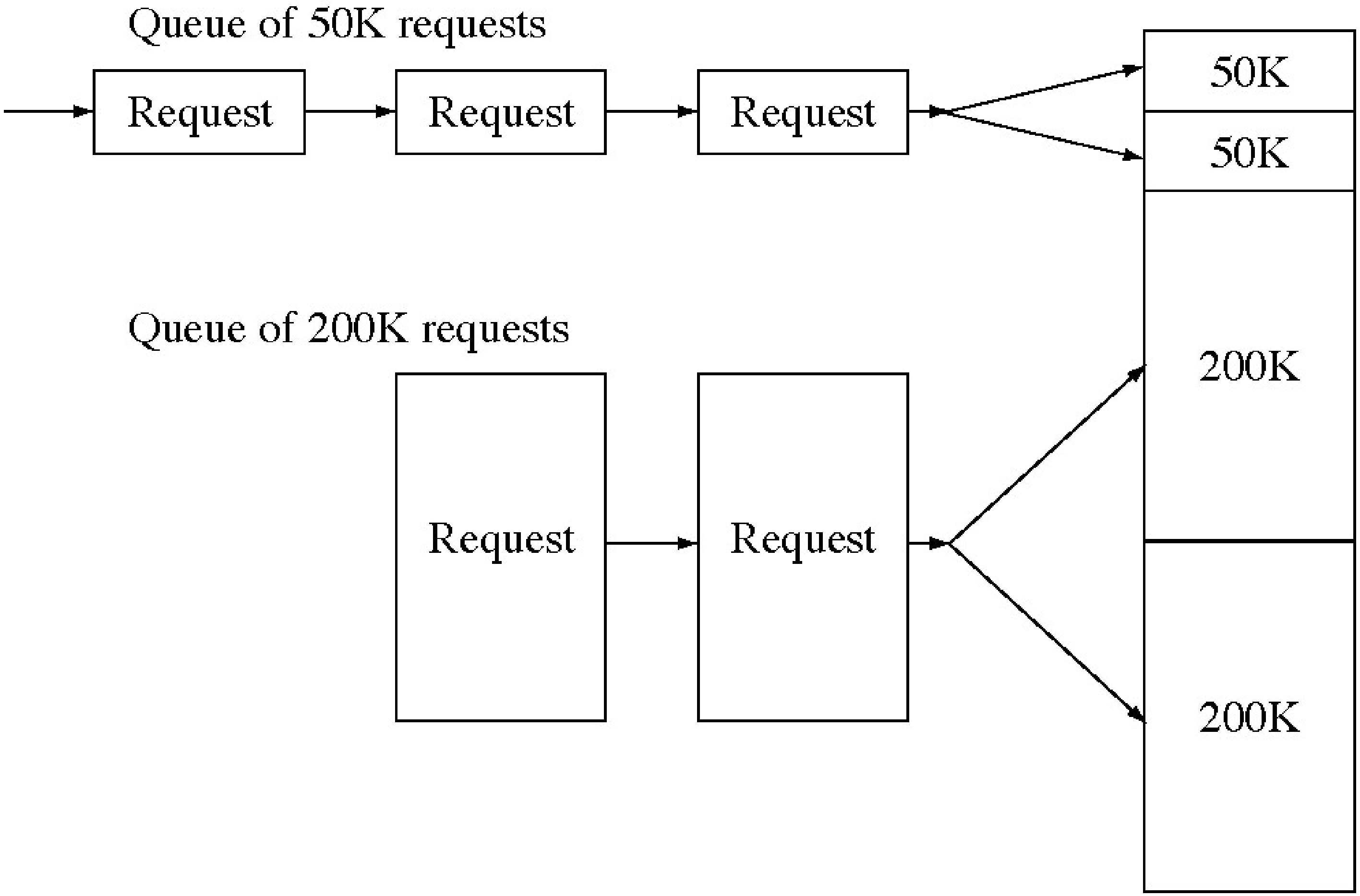
Static and dynamic linking


Method 1:
Static linking
(link time copy)
Library
Executable
Method 2:
Dynamic linking
(load time copy)
Method 3:
Dynamic linking
(run time copy)
Copy in at load time
(always done)
Loaded program
Memory

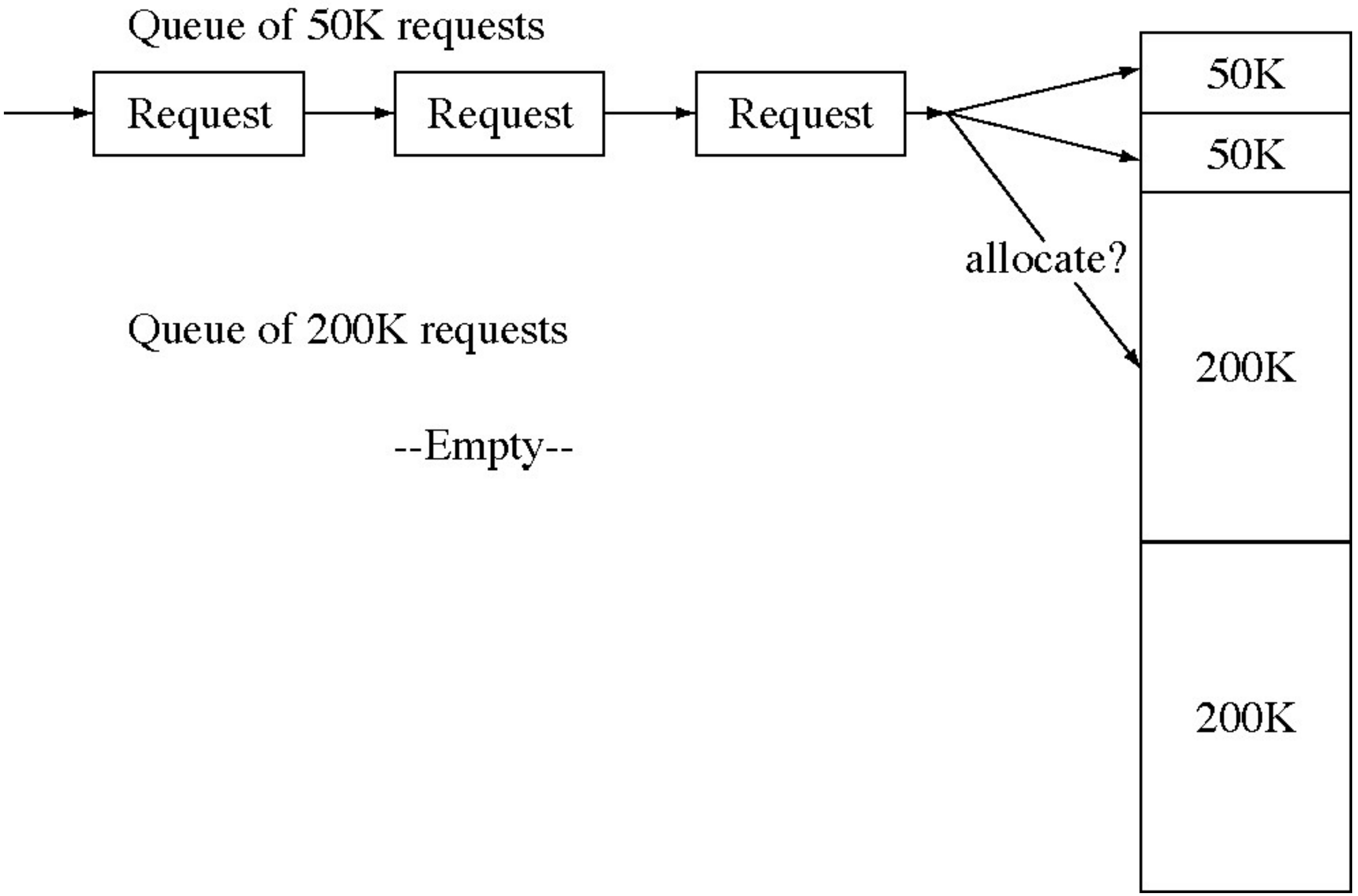
Memory allocation problem


Queue of memory requests
Allocate space
Pool of memory to allocate
Allocated block
New requests enter the system and are placed on a queue
Free block
Completed requests leave the system and free their memory

Queue for each block size

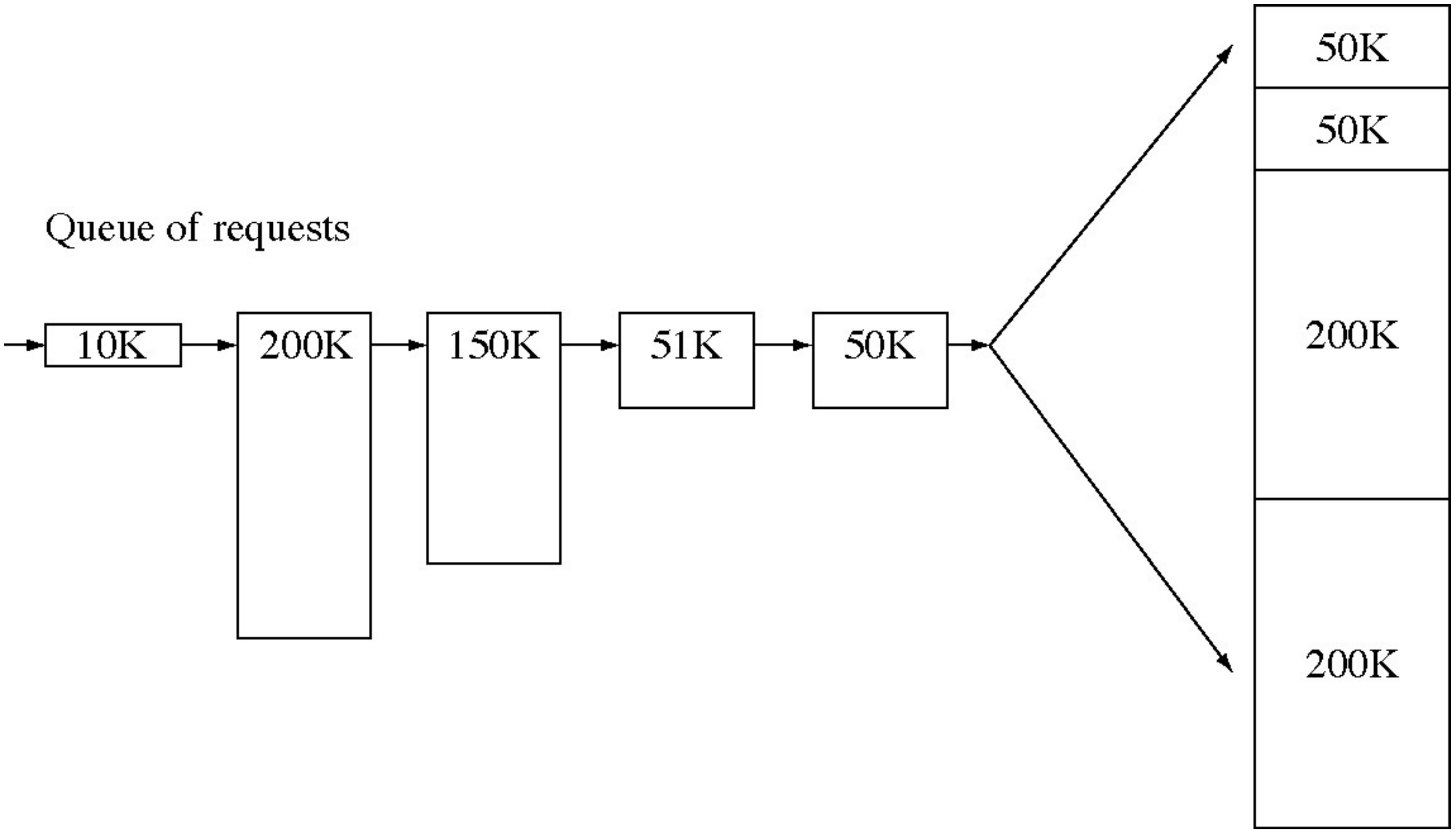

Queue of 50K requests
Request
Request
Request
50K
50K
Queue of 200K requests
Request
Request
200K
200K

Allocate a large block to a small request?



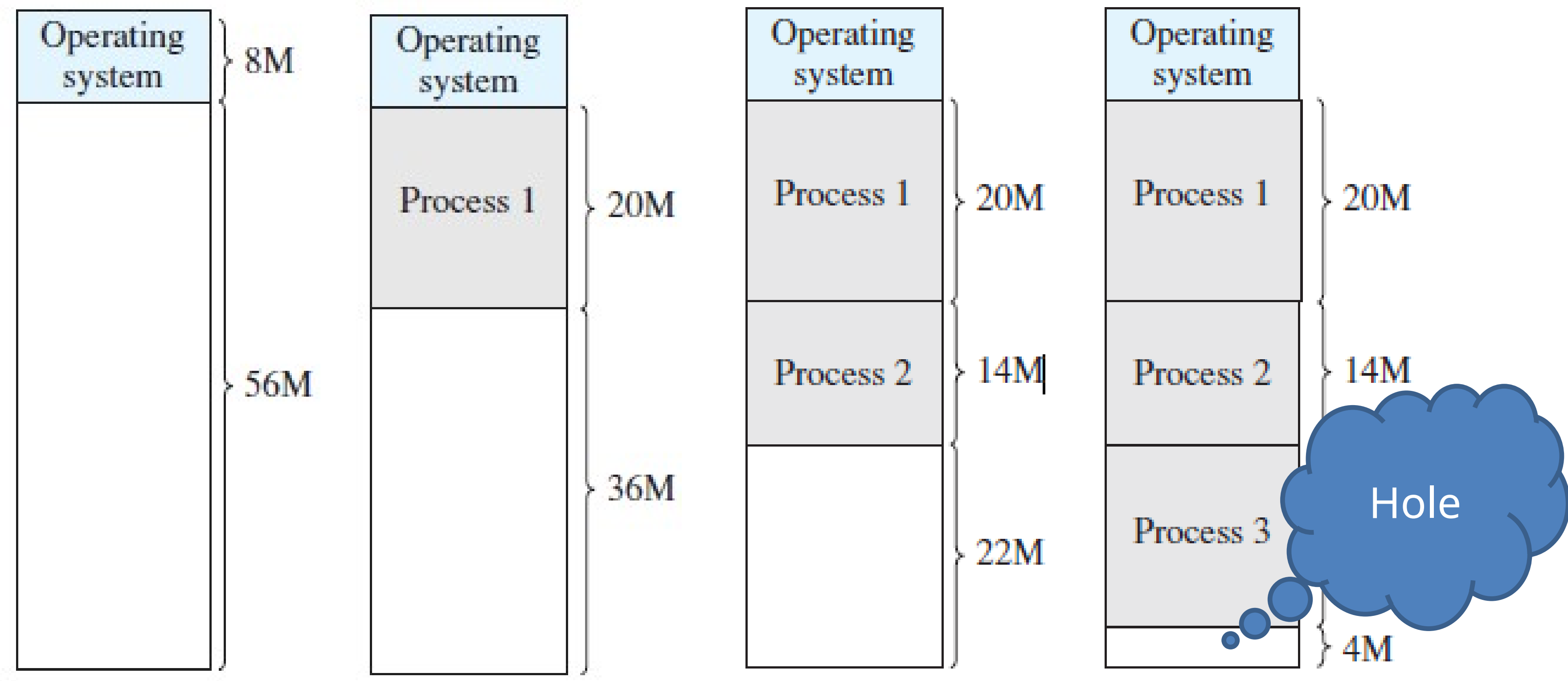
Queue of 50K requests
Request
Request
Request
allocate?
Queue of 200K requests
--Empty--
50K
50K
200K
200K

Variably-sized memory requests

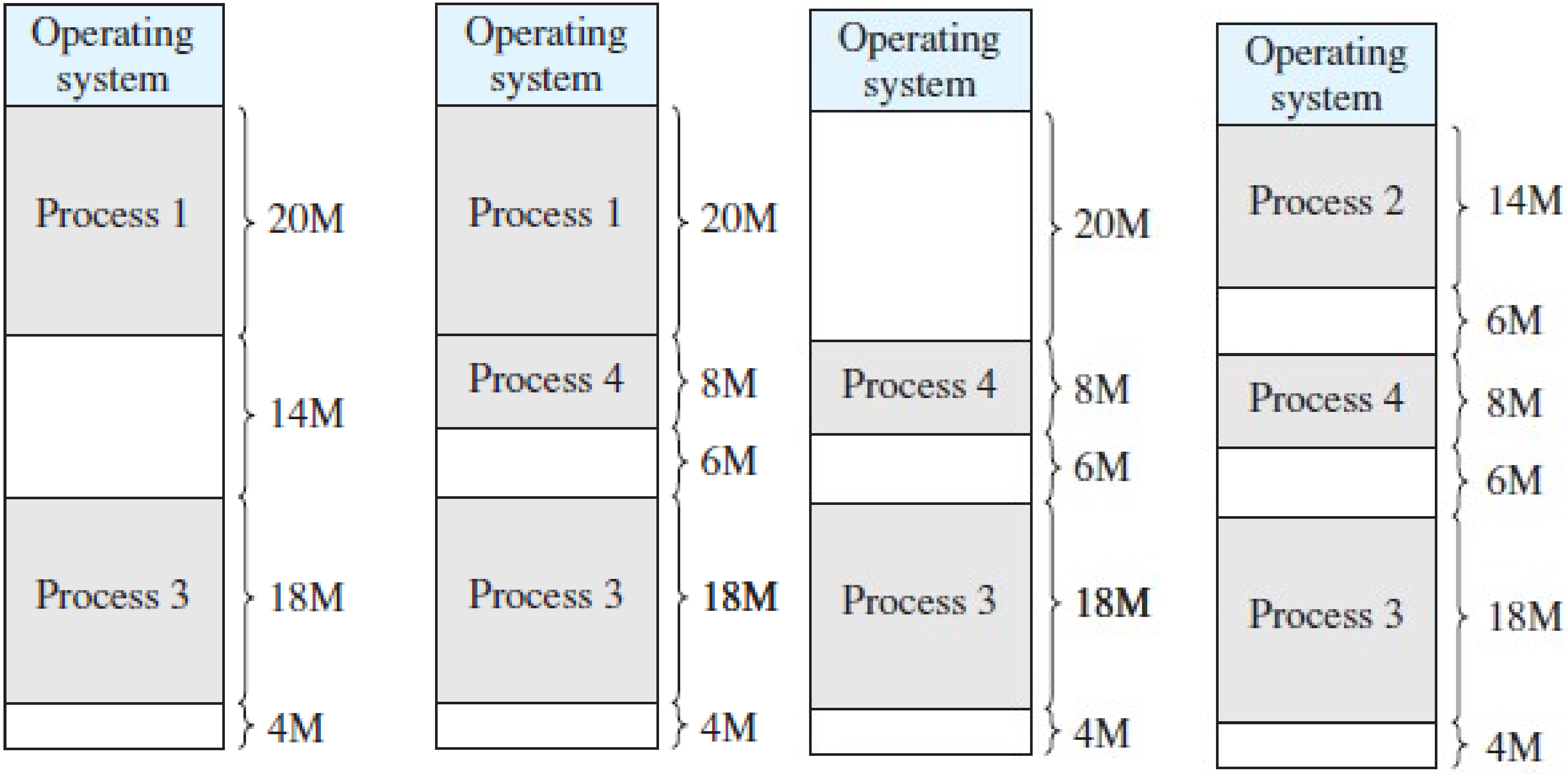

Queue of requests
10K
200K
150K
51K
50K
50K
50K
200K
200K

Dynamic Partitioning or Variable partition allocation

- The partitions are of variable length and number.
- Exact memory allocation


Operating system
8M
56M
Operating system
Process 1
20M
36M
Operating system
Process 1
20M
Process 2
14M
22M
Operating system
Process 1
20M
Process 2
14M
Process 3
4M
Hole

Contd…


Operating system
Process 1
20M
14M
Process 3
18M
4M
Operating system
Process 1
20M
Process 4
8M
6M
Process 3
18M
4M
Operating system
20M
Process 4
8M
6M
Process 3
18M
4M
Operating system
Process 2
14M
6M
Process 4
8M
6M
Process 3
18M
4M

BITS Pilani
Pilani Campus

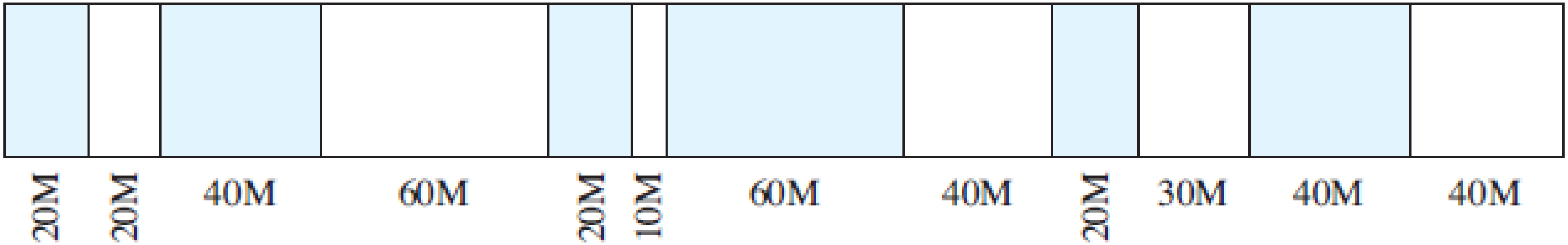
Solution to Memory management design problem & Dynamic memory allocation

Dynamic Storage-Allocation

- Four schemes:
  - **First-fit**: Allocate the *first* hole that is big enough.
  - **Best-fit**: Allocate the *smallest* hole that is big enough; must search entire list, unless ordered by size. Produces the smallest leftover hole.
  - **Next-fit:** begins to scan memory from the location of the last placement, and chooses the next available block that is large enough.
  - **Worst-fit**: Allocate the *largest* hole; must also search entire list. Produces the largest leftover hole.
- Which one is better?

Problem …

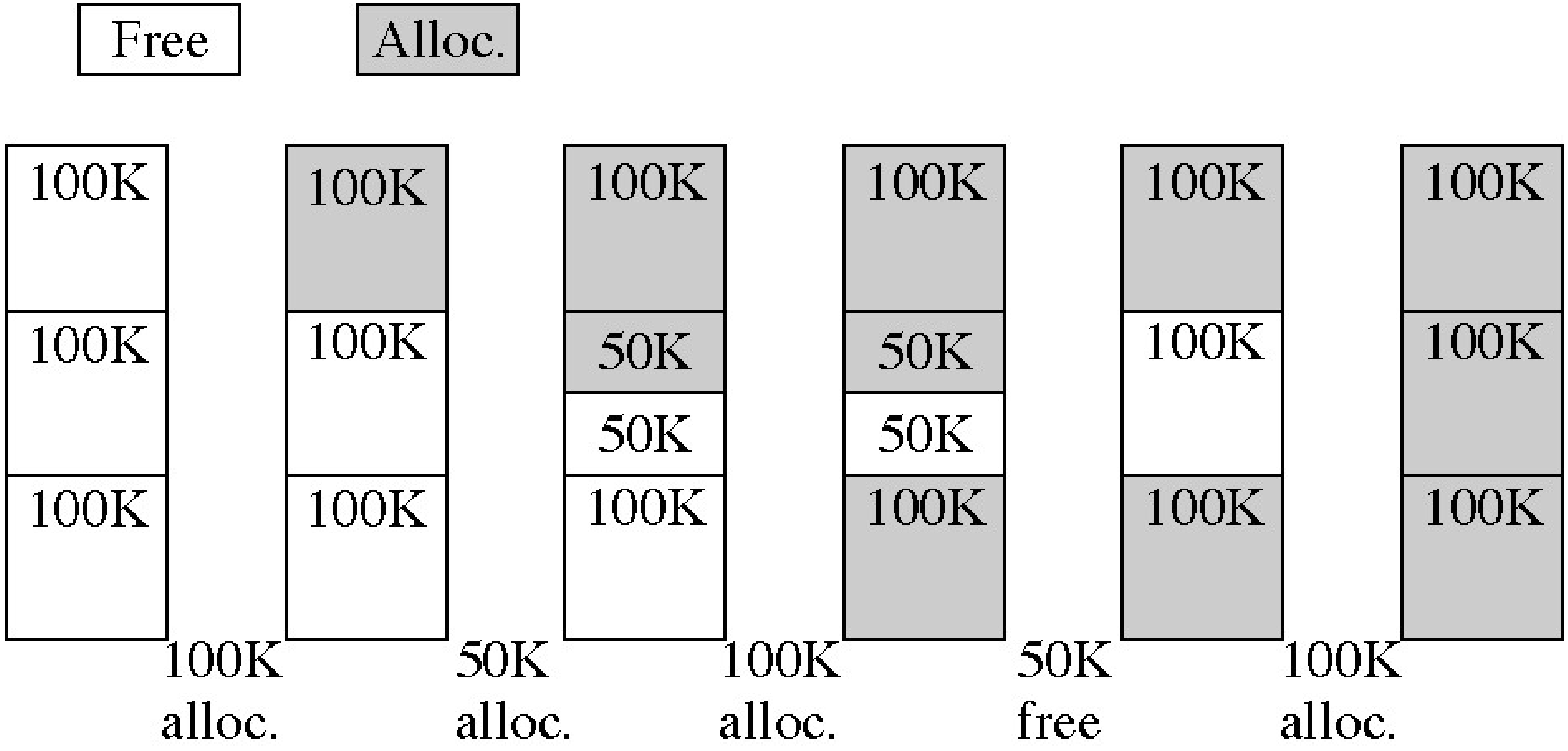
A dynamic partitioning scheme is being used, and the following is the memory configuration at a given point in time:



The shaded areas are allocated blocks; the white areas are free blocks. The next three memory requests are for 40M, 20M, and 10M. Indicate the starting address for each of the three blocks using the following placement algorithms:

a. First-fit
b. Best-fit
c. Next-fit. Assume the most recently added block is at the beginning of memory.
d. Worst-fit

The buddy system


Free
Alloc.
100K
100K
100K
100K alloc.
100K
100K
100K
50K alloc.
100K
50K
50K
100K
100K alloc.
100K
50K
50K
100K
50K free
100K
100K
100K
100K alloc.
100K
100K
100K

Problem ….

Consider a memory block of 16K. Perform the following:

Allocate (A: 3.5K)

Allocate (B: 1.2K)

Allocate (C: 1.3K)

Allocate (D: 1.9K)

Allocate (E: 3.2K)

Free (C)

Free (B)

Allocate (F: 1.6K)

Allocate (G: 1.8K)

Consider a memory block of 16K. Perform the following:

Allocate (A: 3.5K)
Allocate (B: 1.2K)
Allocate (C: 1.3K)
Allocate (D: 1.9K)
Allocate (E: 3.2K)
Free (C)
Free (B)
Allocate (F: 1.6K)
Allocate (G: 1.8K)

- 16K Memory Block

16K

- Allocate (A: 3.5K)

4k (A)	**4K**	**8K**

- Allocate (B: 1.2K)

4k (A)	**2K(B)**	**2K**	**8K**

- Allocate (C: 1.3K)

4k (A)	**2K(B)**	**2K(C)**	**8K**

- Allocate (D: 1.9K)

4k (A)	**2K(B)**	**2K(C)**	**2K(D)**	**2K**	**4K**

Consider a memory block of 16K. Perform the following:

Allocate (A: 3.5K)
Allocate (B: 1.2K)
Allocate (C: 1.3K)
Allocate (D: 1.9K)
Allocate (E: 3.2K)
Free (C)
Free (B)
Allocate (F: 1.6K)
Allocate (G: 1.8K)

- Allocate (E: 3.2K)

4k (A)	2K(B)	2K(C)	2K(D)	2K	4K(E)

- Free (C)

4k (A)	2K(B)	2K	2K(D)	2K	4K(E)

- Free (B)

4k (A)	2K	2K	2K(D)	2K	4K(E)

4k (A)	4K	2K(D)	2K	4K(E)

- Allocate (F: 1.6K)

4k (A)	4k	2K(D)	2K(F)	4K(E)

- Allocate (G: 1.8K)

4k (A)	2K(G)	2K	2K(D)	2K(F)	4K(E)

A system has 1024 KB of memory, and it uses the Buddy System for memory allocation. The following memory allocation and deallocation requests occur:

Process A requests 200 KB of memory.

Process B requests 350 KB of memory.

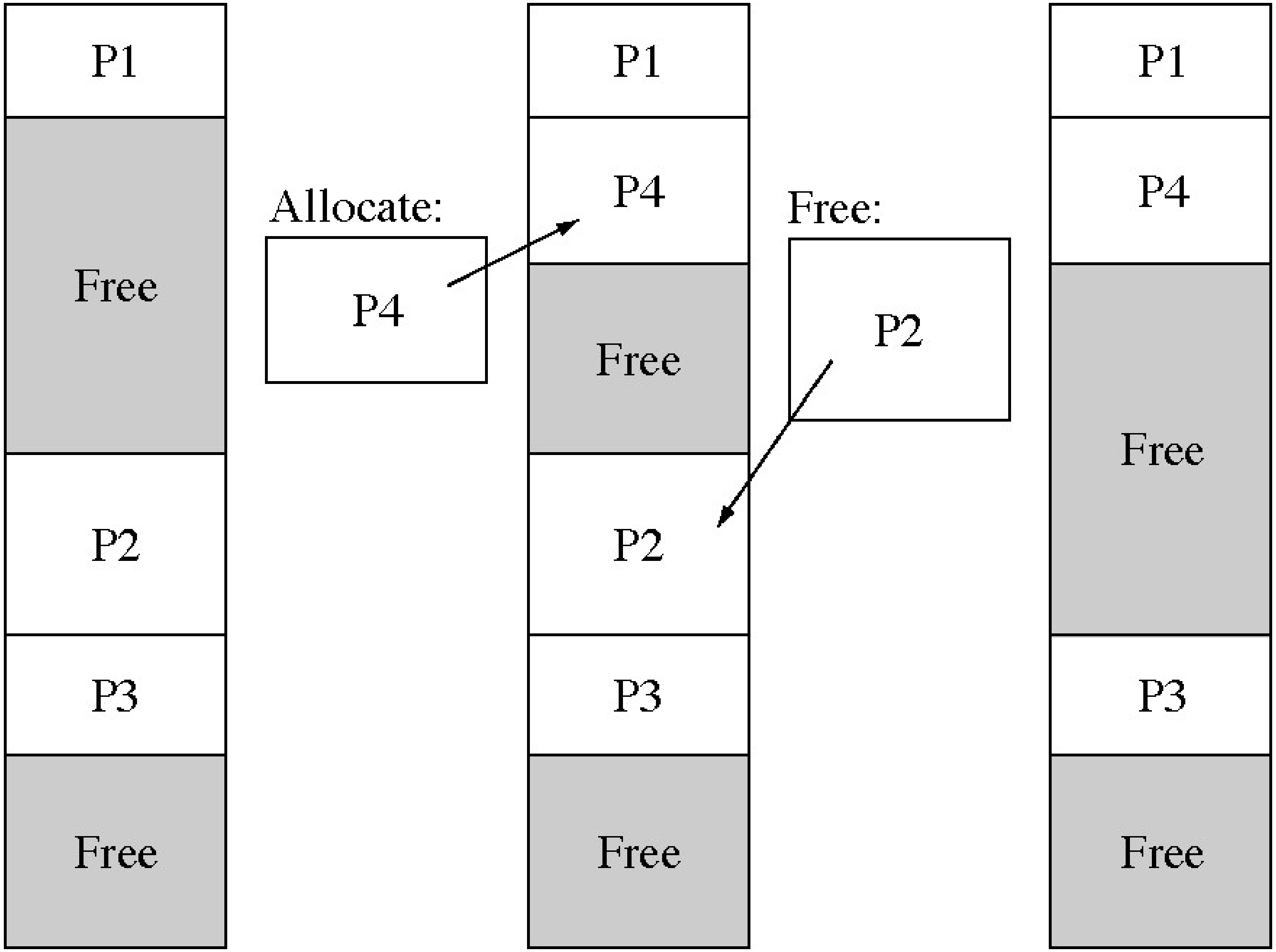
Process C requests 100 KB of memory.

Process A releases its memory.

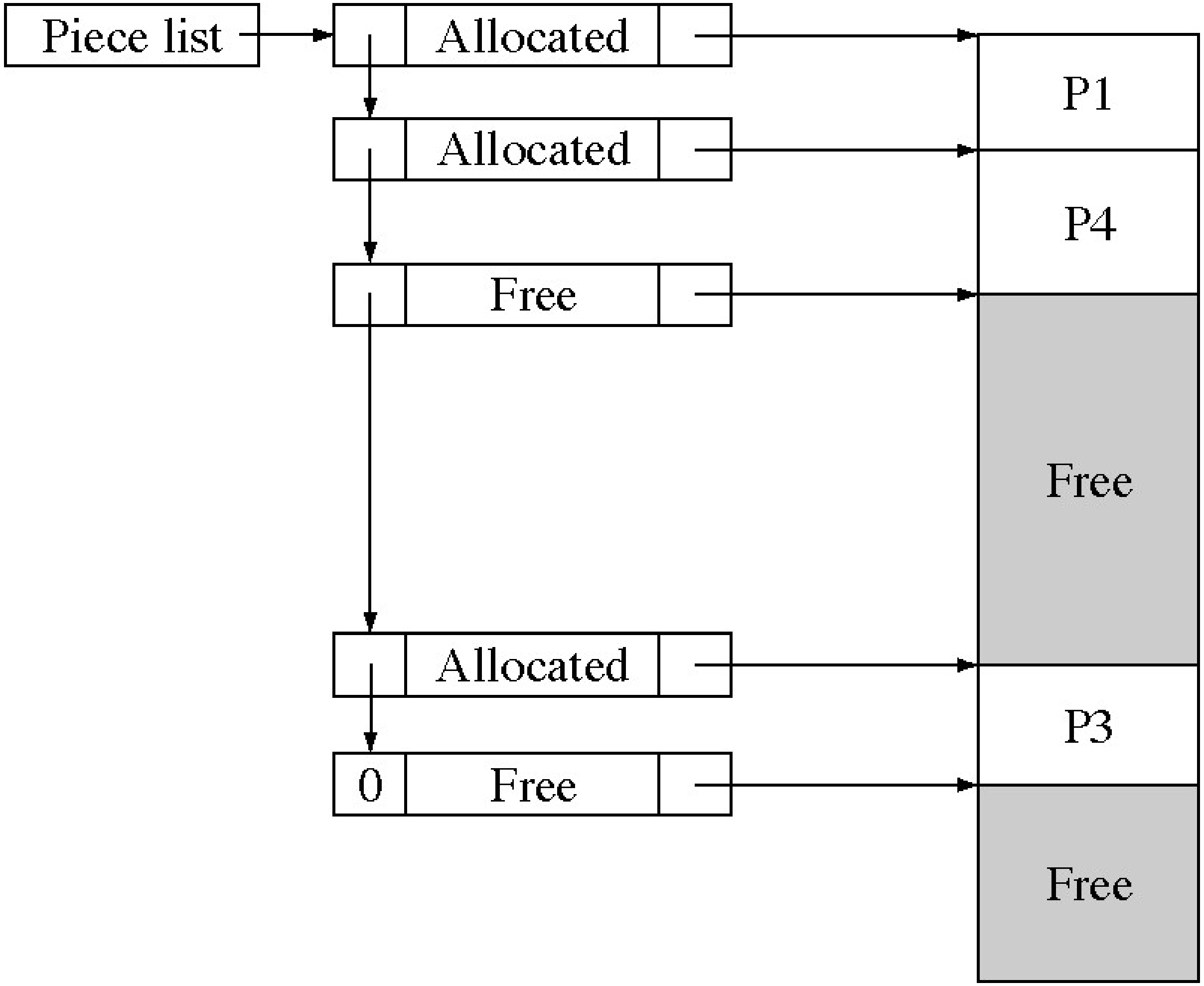
Process D requests 250 KB of memory.

Process C releases its memory.

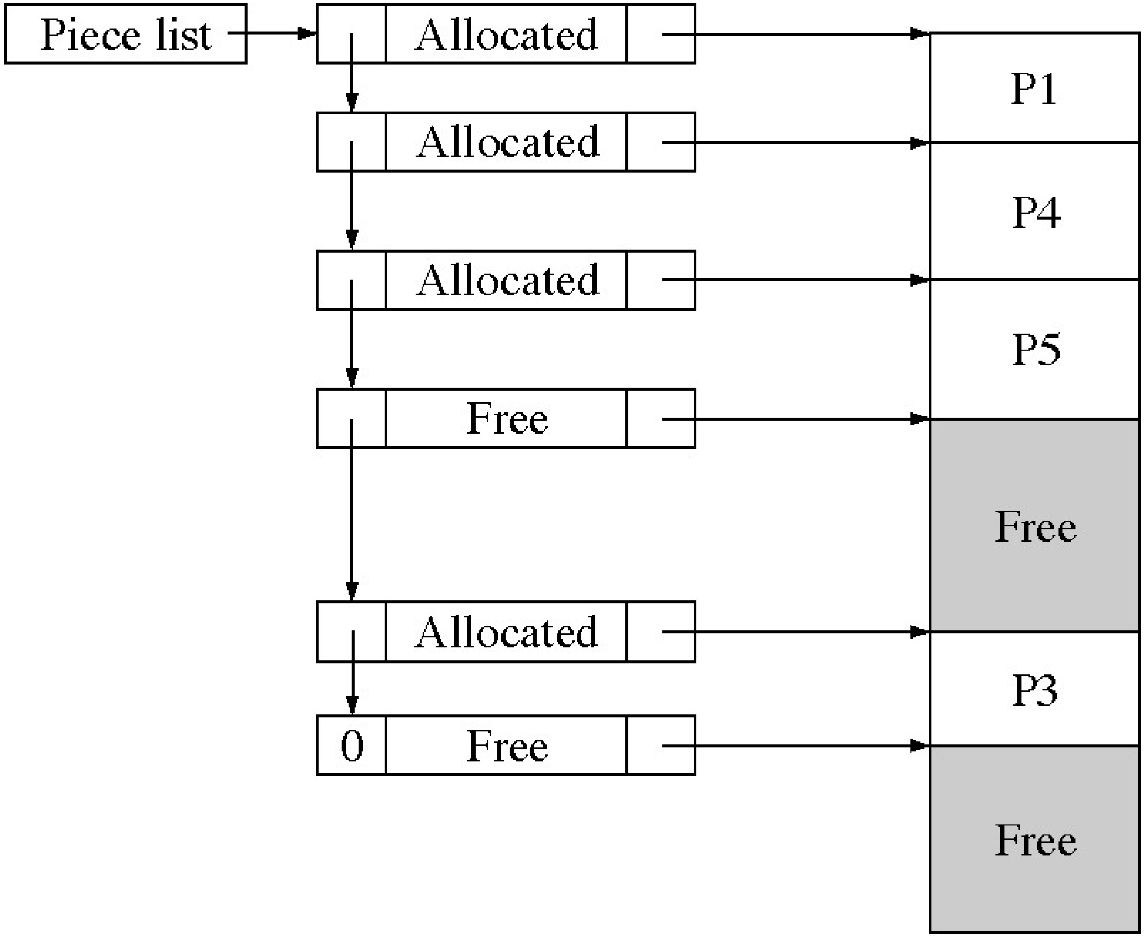
Allocating and freeing blocks


P1
Free
P2
P3
Free
Allocate:
P4
P1
P4
Free
P2
P3
Free
Free:
P2
P1
P4
Free
P3
Free

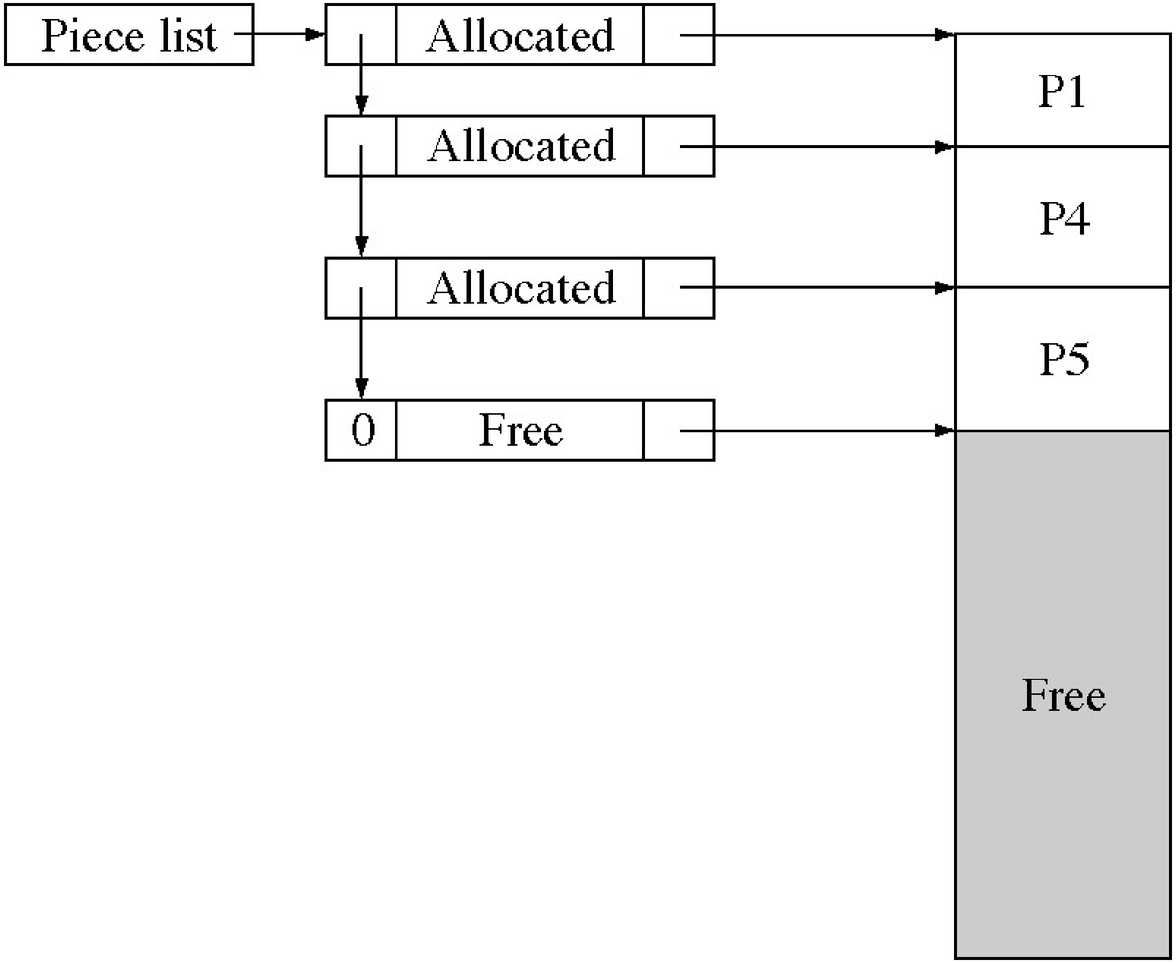
The block list method


Piece list
Allocated
Allocated
Free
Allocated
0
Free
P1
P4
Free
P3
Free

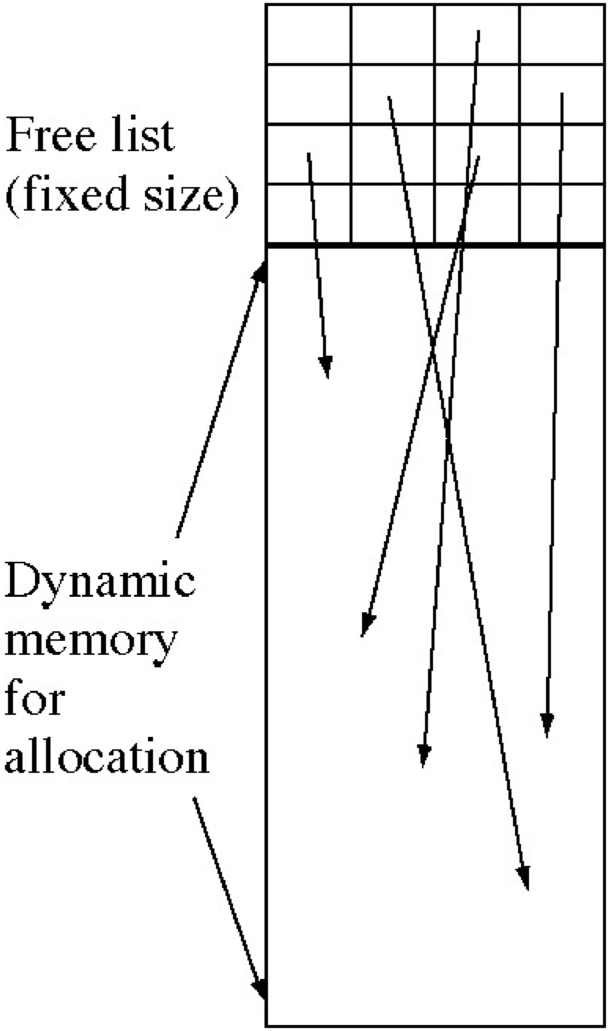
After allocating P5


Piece list
Allocated
Allocated
Allocated
Free
Allocated
0
Free
P1
P4
P5
Free
P3
Free

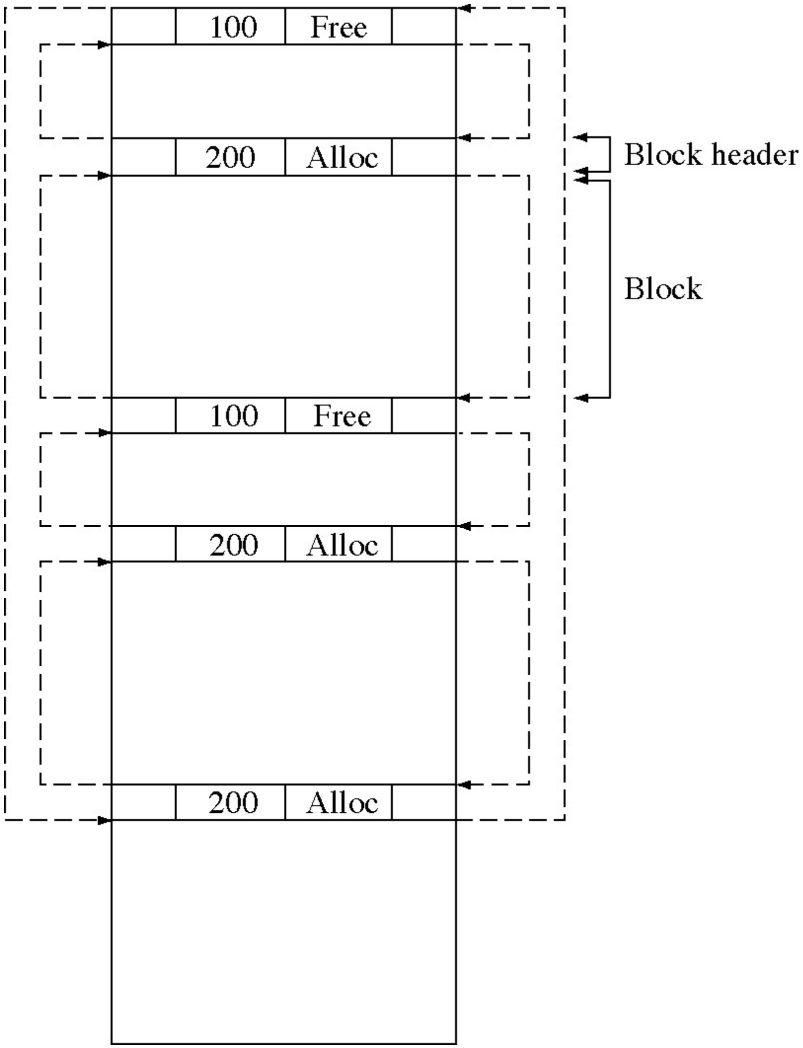
After freeing P3


Piece list
Allocated
Allocated
Allocated
0
Free
P1
P4
P5
Free

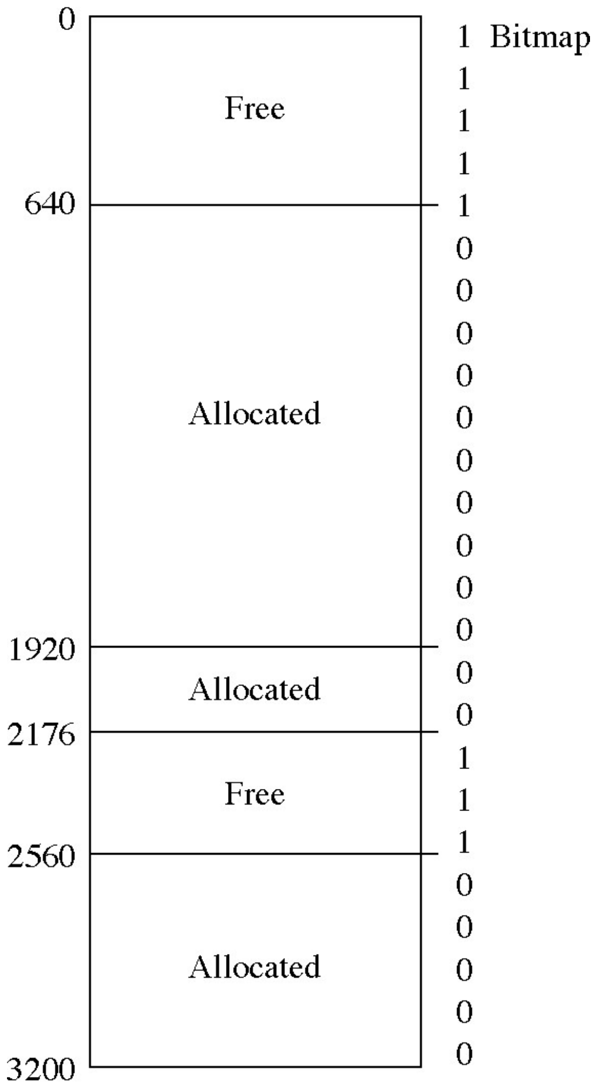
Reserving space for the block list


Free list
(fixed size)
Dynamic
memory
for
allocation

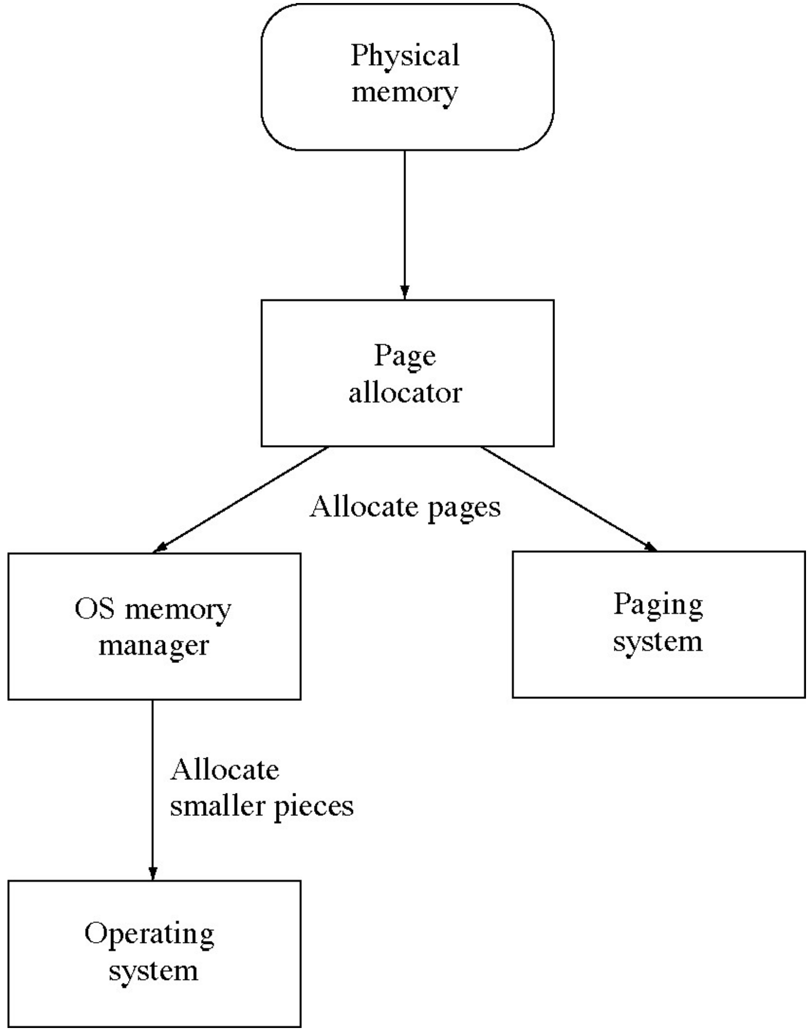
Block list with list headers


100
Free
200
Alloc
100
Free
200
Alloc
200
Alloc
Block header
Block

Bitmap method


0
640
1920
2176
2560
3200
Free
Allocated
Allocated
Free
Allocated
Bitmap
1
1
1
1
1
0
0
0
0
0
0
0
0
0
0
0
0
1
1
1
0
0
0
0
0

Allocating memory in a paging system

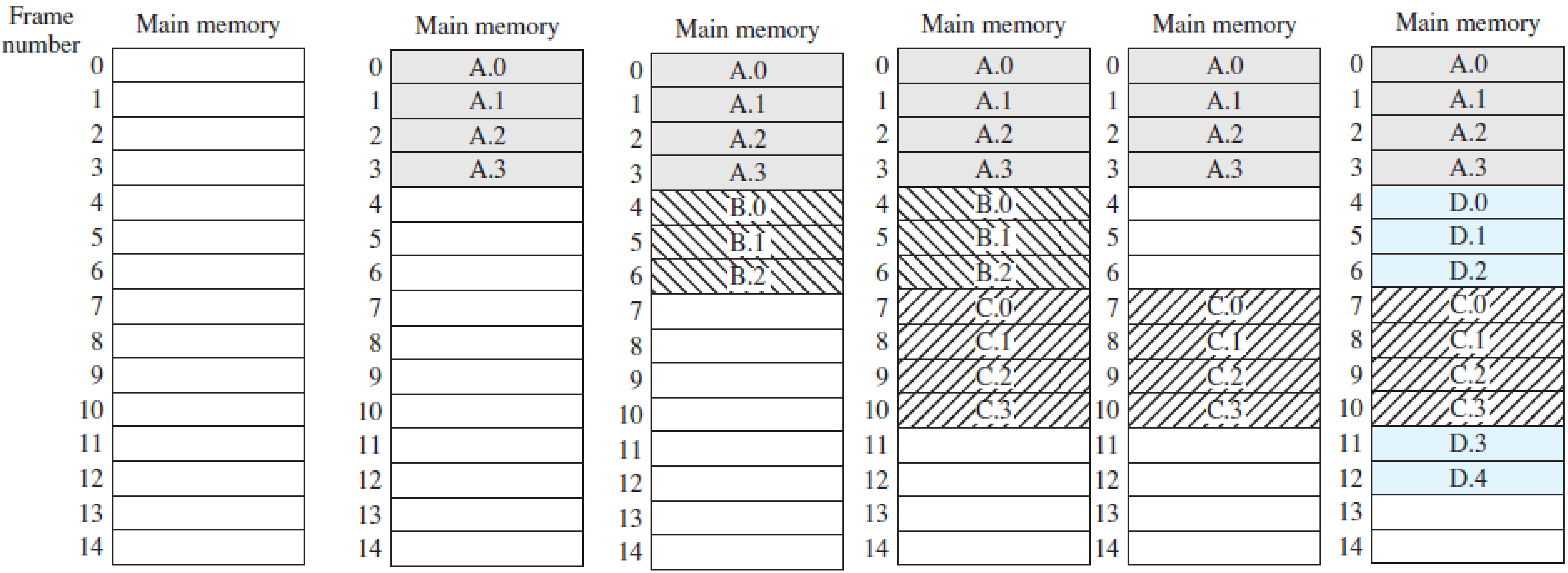

Physical memory
Page allocator
Allocate pages
OS memory manager
Paging system
Allocate smaller pieces
Operating system

Paging

- Paging permits the physical address space of a process to be non- contiguous.
- Divide physical memory into fixed-sized blocks called **frames** (size is power of 2).
- Divide logical memory into blocks of same size called **pages**.
- Keep track of all free frames.
- To run a program of size *n* pages, need to find *n* free frames and load program.
- Keep track of all free frames.
- Set up a page table to translate logical to physical

Example

Process A : 4 pages
Process B : 3 pages
Process C : 4 Pages
Process D : 5 Pages
Main Memory : 15 frames



Frame number	(a) Fifteen available frames	(b) Load process A	(c) Load process B	(d) Load process C	(e) Swap out B	(f) Load process D
0		A.0	A.0	A.0	A.0	A.0
1		A.1	A.1	A.1	A.1	A.1
2		A.2	A.2	A.2	A.2	A.2
3		A.3	A.3	A.3	A.3	A.3
4			B.0	B.0		D.0
5			B.1	B.1		D.1
6			B.2	B.2		D.2
7				C.0	C.0	C.0
8				C.1	C.1	C.1
9				C.2	C.2	C.2
10				C.3	C.3	C.3
11						D.3
12						D.4
13						
14						

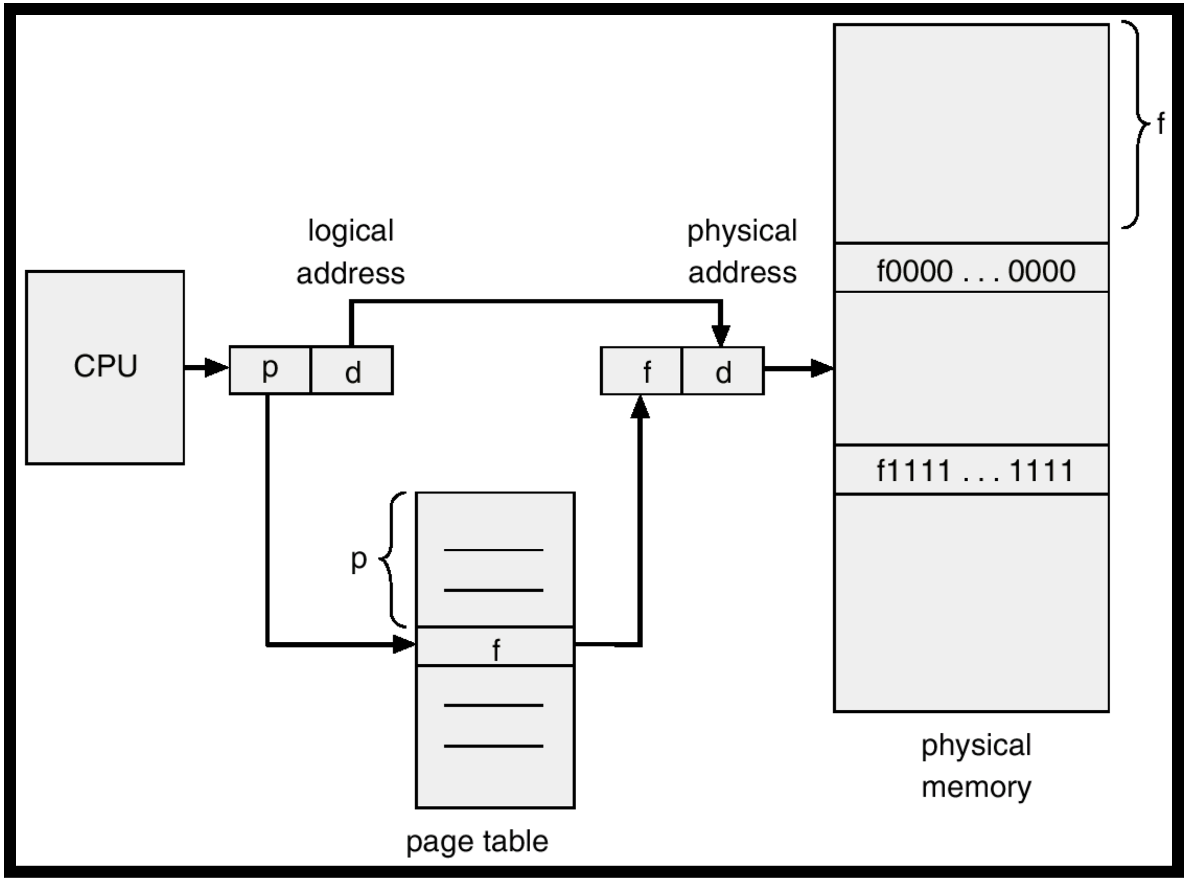
(a) Fifteen available frames (b) Load process A (c) Load process B (d) Load process C (e) Swap out B (f) Load process D

Address Translation Scheme

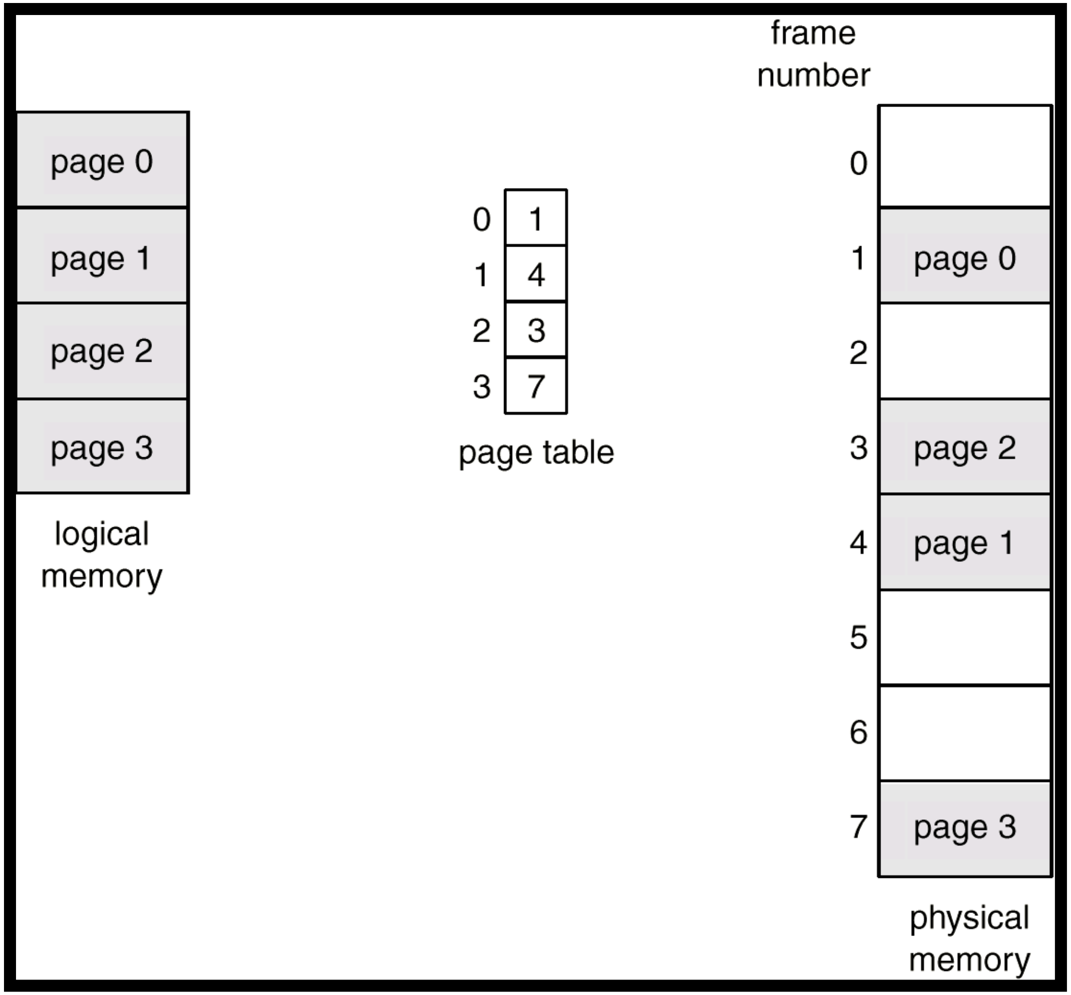
- Address generated by CPU is divided into:
    - *Page number (p)* – used as an index into a *page table.* Page table contains base address of each page in physical memory.
    - *Page offset (d)* – combined with base address to define the physical memory address that is sent to the memory unit.
    - For given logical address space 2^m *and page size* 2^n

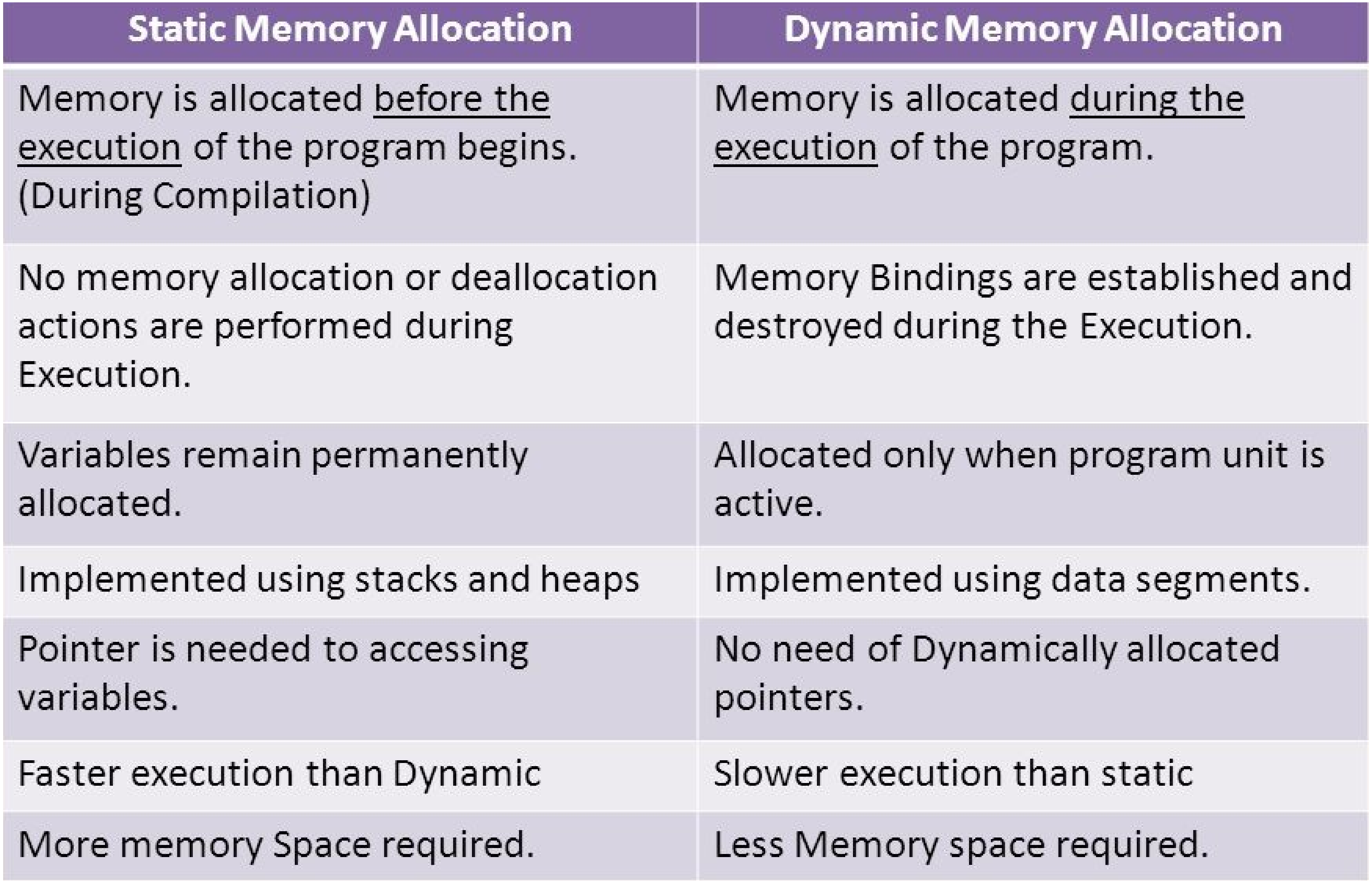
page number	page offset
p	*d*
m - n	*n*

Address Translation Architecture


logical
address
physical
address
CPU
p
d
f
d
p
f
page table
f0000 . . . 0000
f1111 . . . 1111
f
physical
memory

Paging Example


frame
number
page 0
page 1
page 2
page 3
logical
memory
0 1
1 4
2 3
3 7
page table
0
1 page 0
2
3 page 2
4 page 1
5
6
7 page 3
physical
memory



Static Memory Allocation	Dynamic Memory Allocation
Memory is allocated before the execution of the program begins. (During Compilation)	Memory is allocated during the execution of the program.
No memory allocation or deallocation actions are performed during Execution.	Memory Bindings are established and destroyed during the Execution.
Variables remain permanently allocated.	Allocated only when program unit is active.
Implemented using stacks and heaps	Implemented using data segments.
Pointer is needed to accessing variables.	No need of Dynamically allocated pointers.
Faster execution than Dynamic	Slower execution than static
More memory Space required.	Less Memory space required.

THANK YOU

BITS Pilani
Pilani Campus

COMPUTING AND DESIGN

SESSION 8

Course design by - Dr. Lucy J. Gudino &
Dr Chandrasekhar
Faculty – Balamurali Shankar
WILP & Department of CS & IS

BITS Pilani
Pilani Campus

Operating systems – Virtual Memory Design

BITS Pilani
Pilani Campus

Last Session

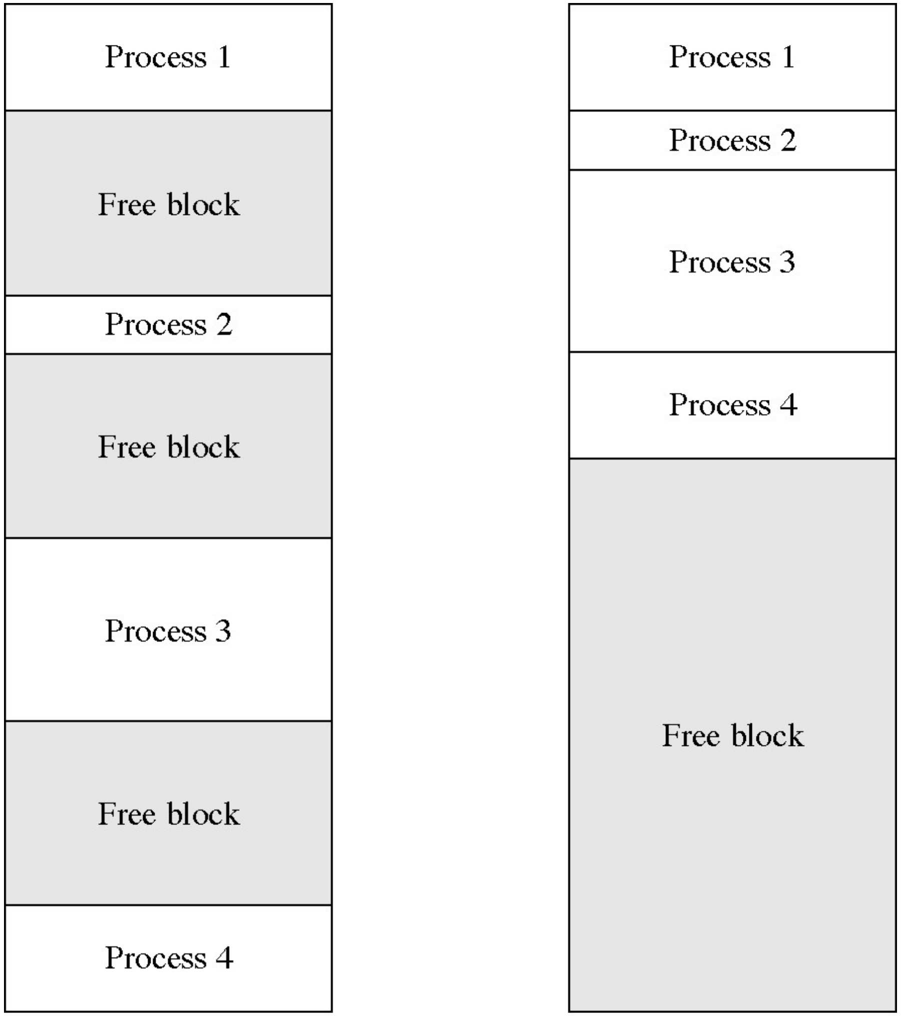
List of Topic Title	Text/Ref Book/external resource
Levels of memory management	T1 (10.1)
The memory management design problem	T1 (10.5)
Solution to the memory management design problem	T1(10.6)
Dynamic memory allocation	T1(10.7)

Virtual Memory Design

Today's topic

- Fragmentation and compaction
- Page table location : Hardware register Vs. Main memory Vs Cache
- Implementing Virtual memory

Fragmentation / Compaction



Process 1
Free block
Process 2
Free block
Process 3
Free block
Process 4
Process 1
Process 2
Process 3
Process 4
Free block

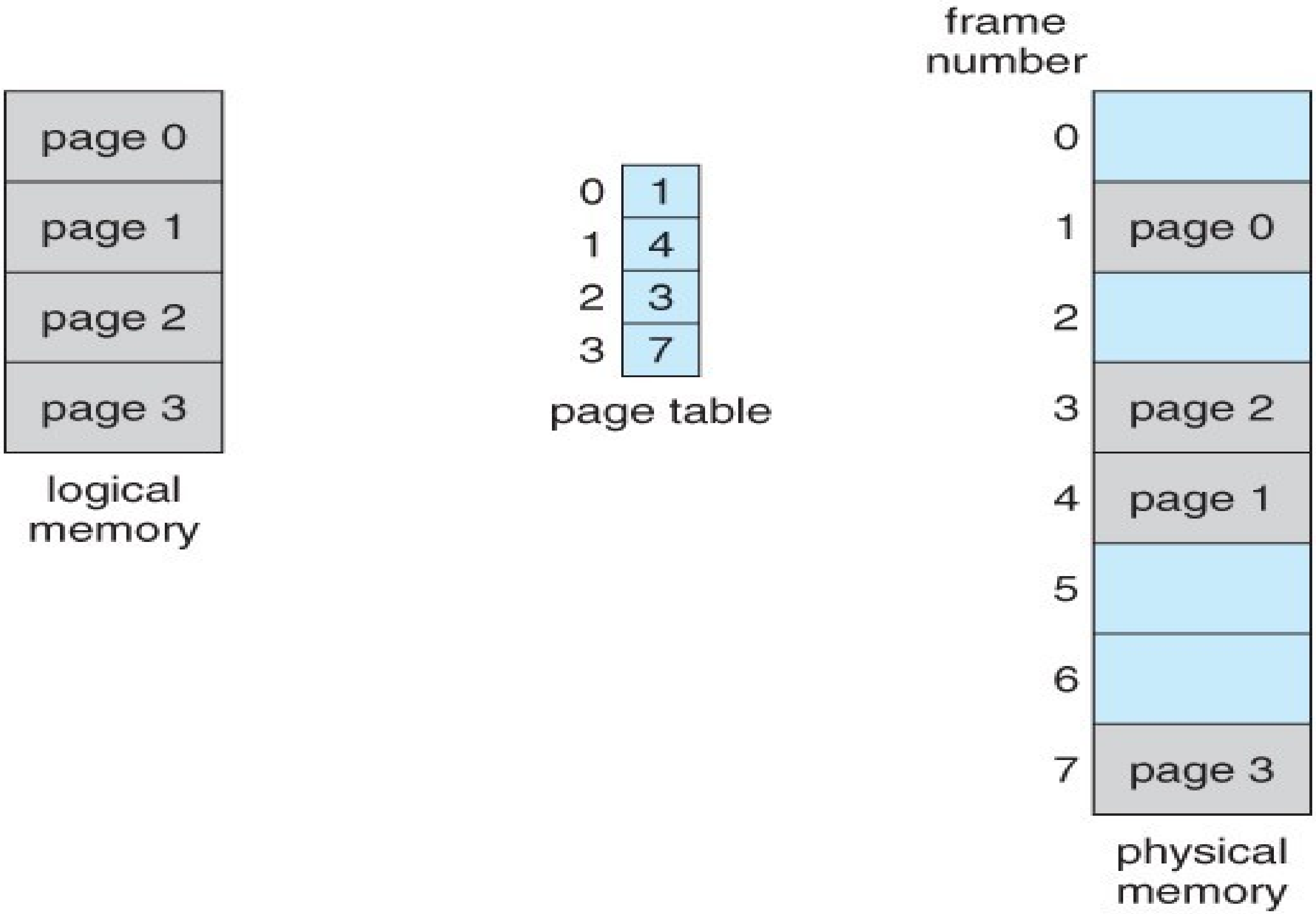
Before compaction

After compaction

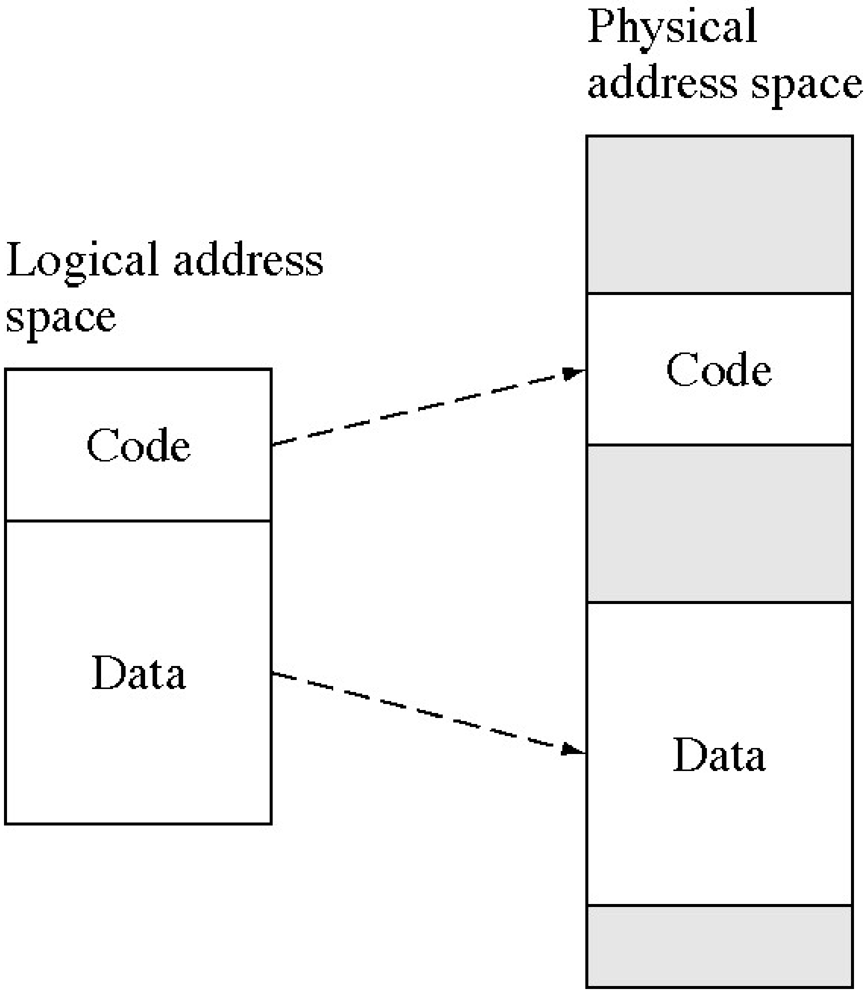
Fragmentation / Compaction

- Without memory mapping, programs require physically continuous memory
- Large blocks mean large fragments
  - and wasted memory
- We need hardware memory mapping to address this problem
  - segments
  - pages
- We will look at a series of potential solutions

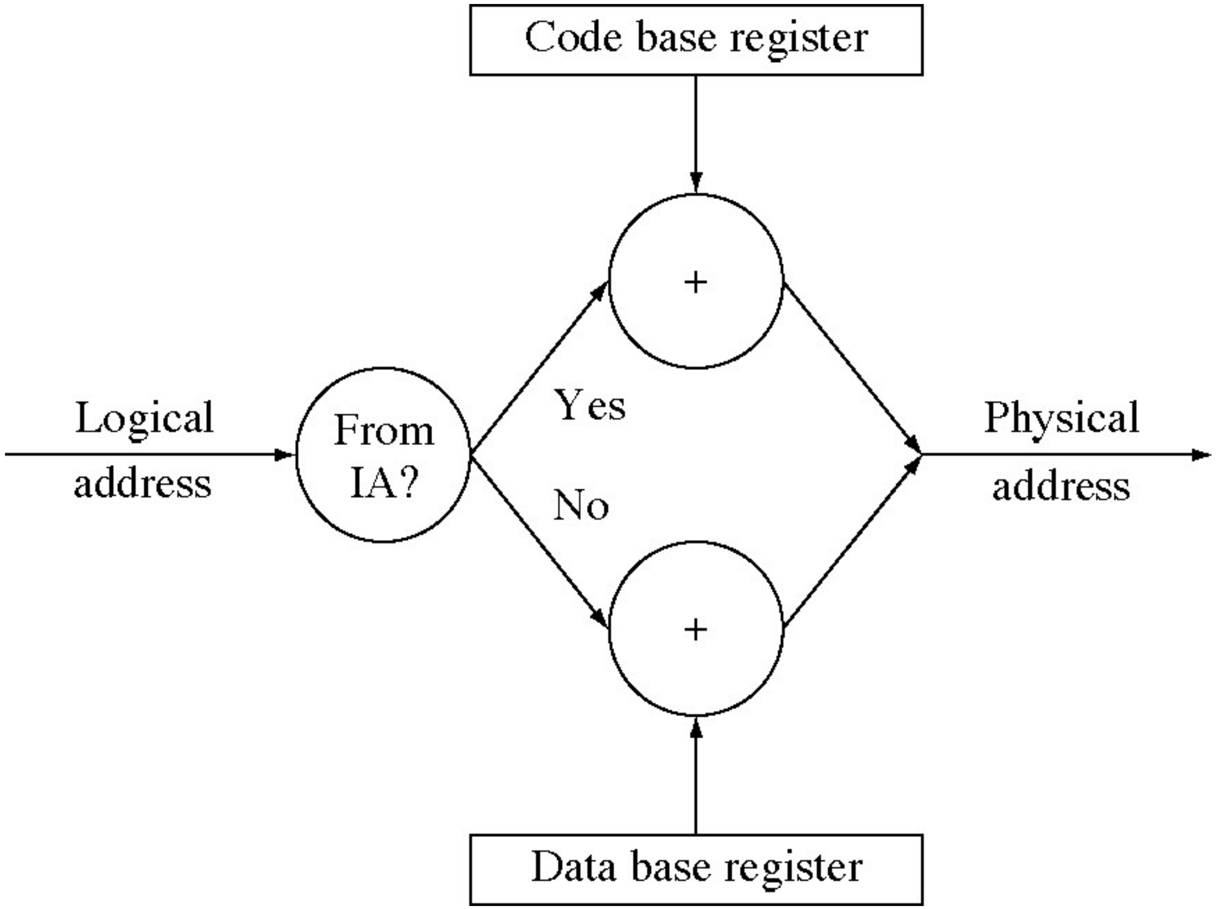
Paging


page 0
page 1
page 2
page 3
logical memory
0
1
2
3
1
4
3
7
page table
frame number
0
1
2
3
4
5
6
7
page 0
page 2
page 1
page 3
physical memory

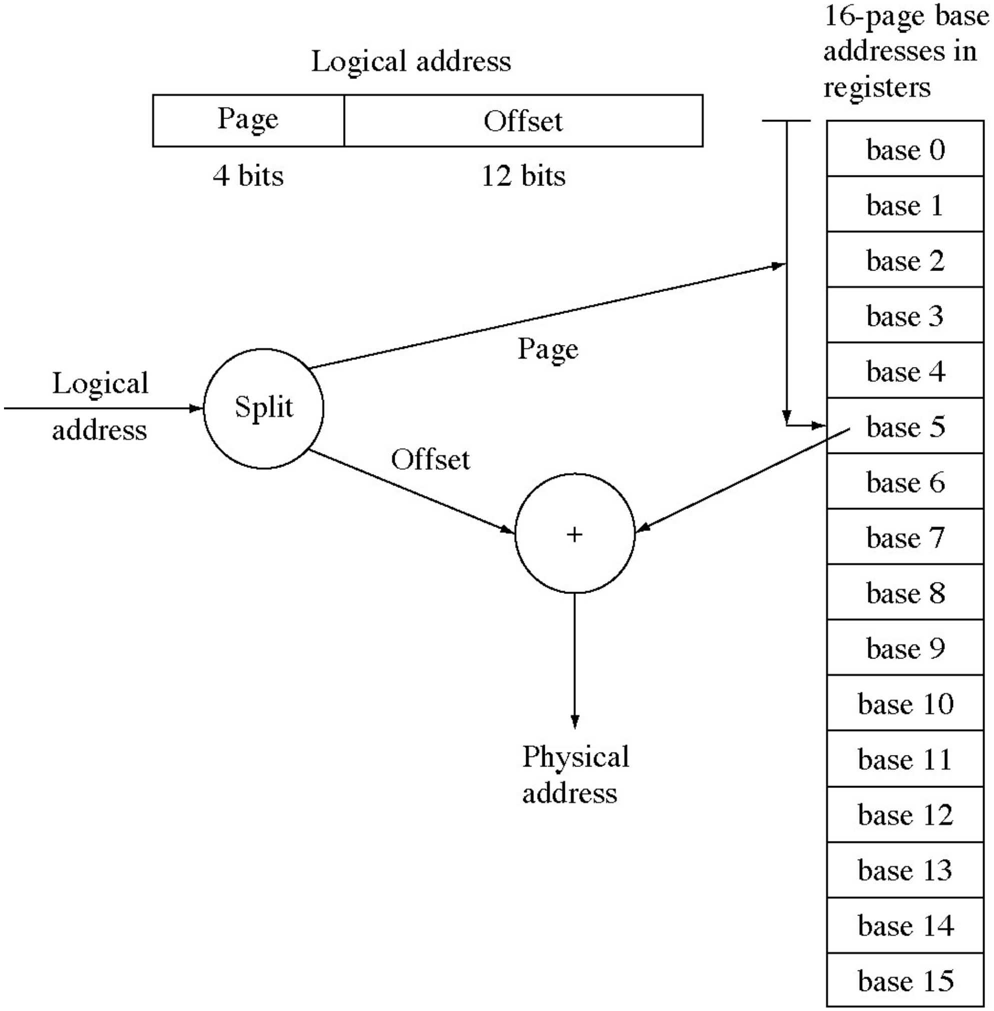
Separate code and data spaces


Physical
address space
Logical address
space
Code
Data
Code
Data

Code/data memory relocation


Code base register
+
Logical address
From IA?
Yes
No
Physical address
+
Data base register

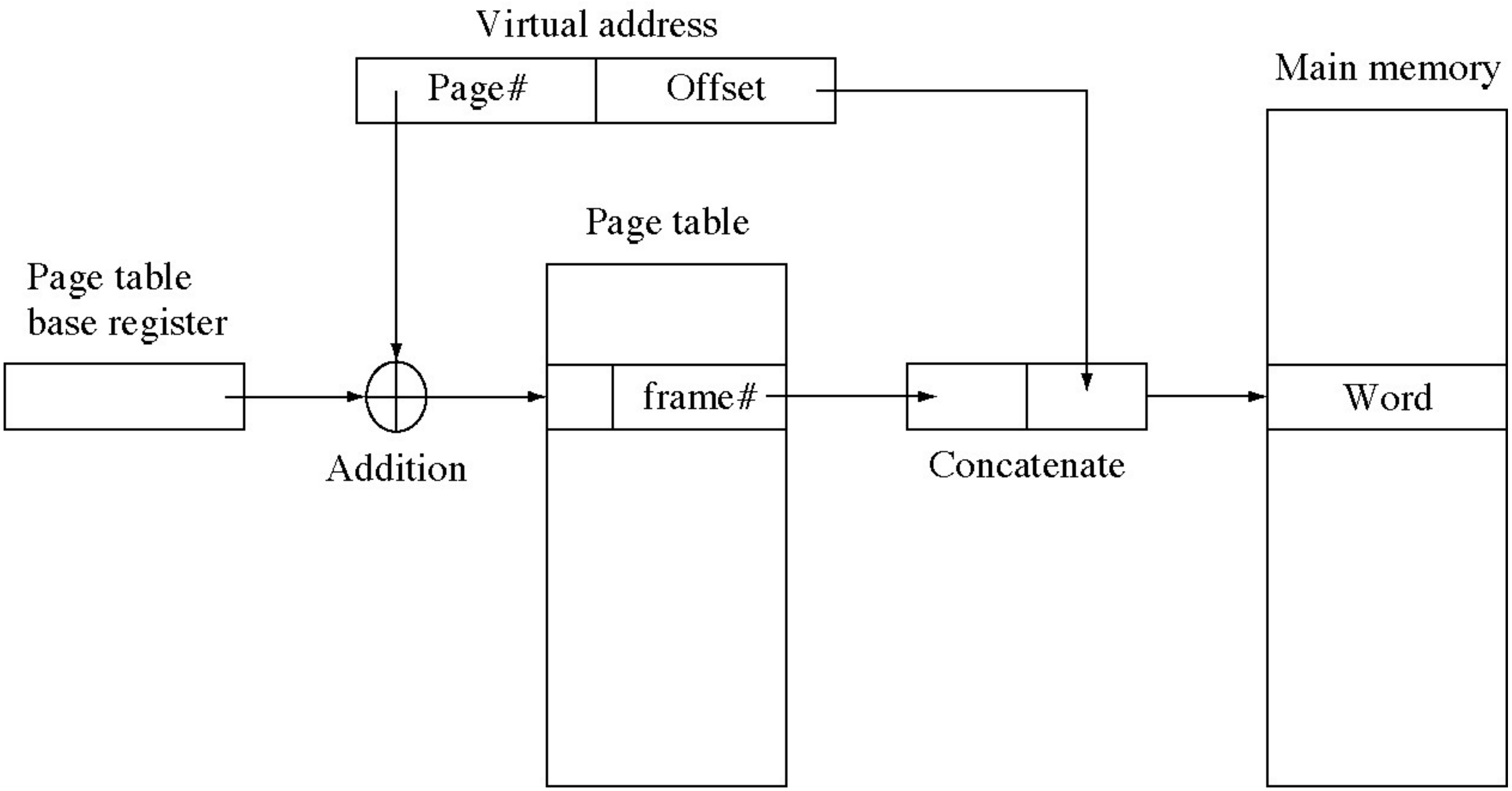
Hardware register page table


Logical address
Page
Offset
4 bits
12 bits
16-page base addresses in registers
base 0
base 1
base 2
base 3
base 4
base 5
base 6
base 7
base 8
base 9
base 10
base 11
base 12
base 13
base 14
base 15
Logical address
Split
Page
Offset
+
Physical address

Problems with page tables in registers

- Practical limit on the number of pages
- Time to save and load page registers on context switches
- Cost of hardware registers
- *Solution*: put the page table in memory and have a single register that points to it

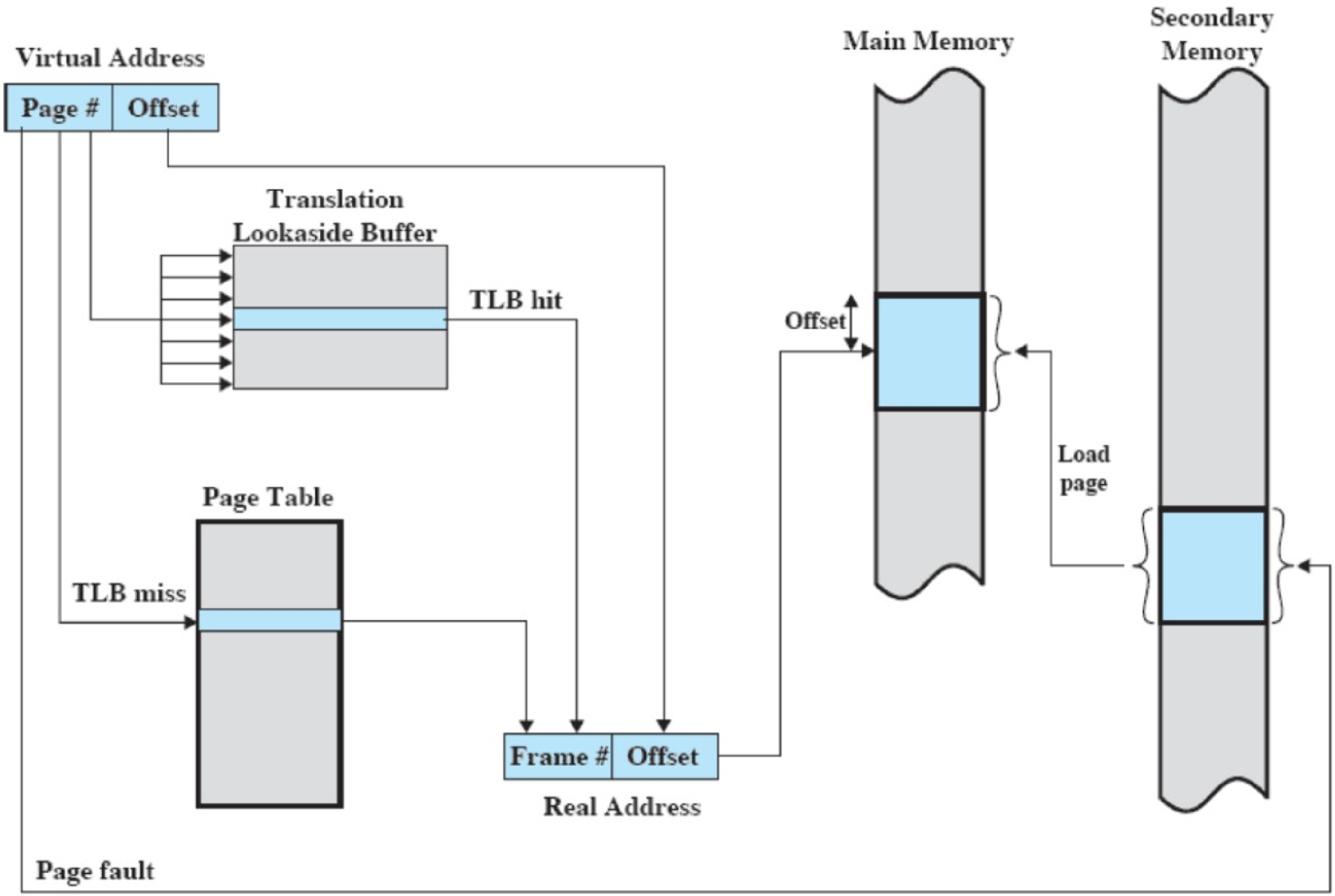
Page table mapping


Virtual address
Page#
Offset
Main memory
Page table
Page table
base register
frame#
Word
Addition
Concatenate

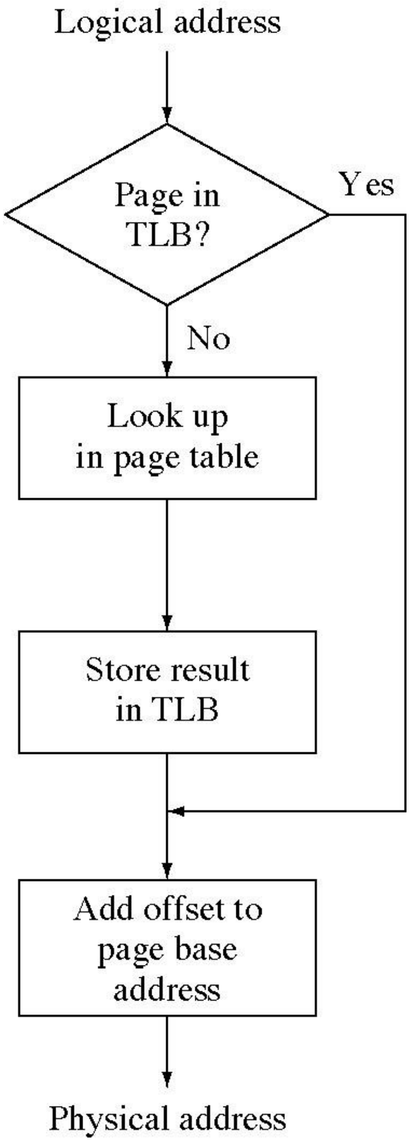
Problems with page tables in memory

- Every data memory access requires a corresponding page table memory access
  - the memory usage has doubled
  - and program speed is cut in half
- *Solution*: caching page table entries
  - called a *translation lookaside buffer*
  - or TLB

Translation look aside buffer


Virtual Address
Page #
Offset
Translation Lookaside Buffer
TLB hit
Page Table
TLB miss
Frame #
Offset
Real Address
Page fault
Main Memory
Offset
Load page
Secondary Memory

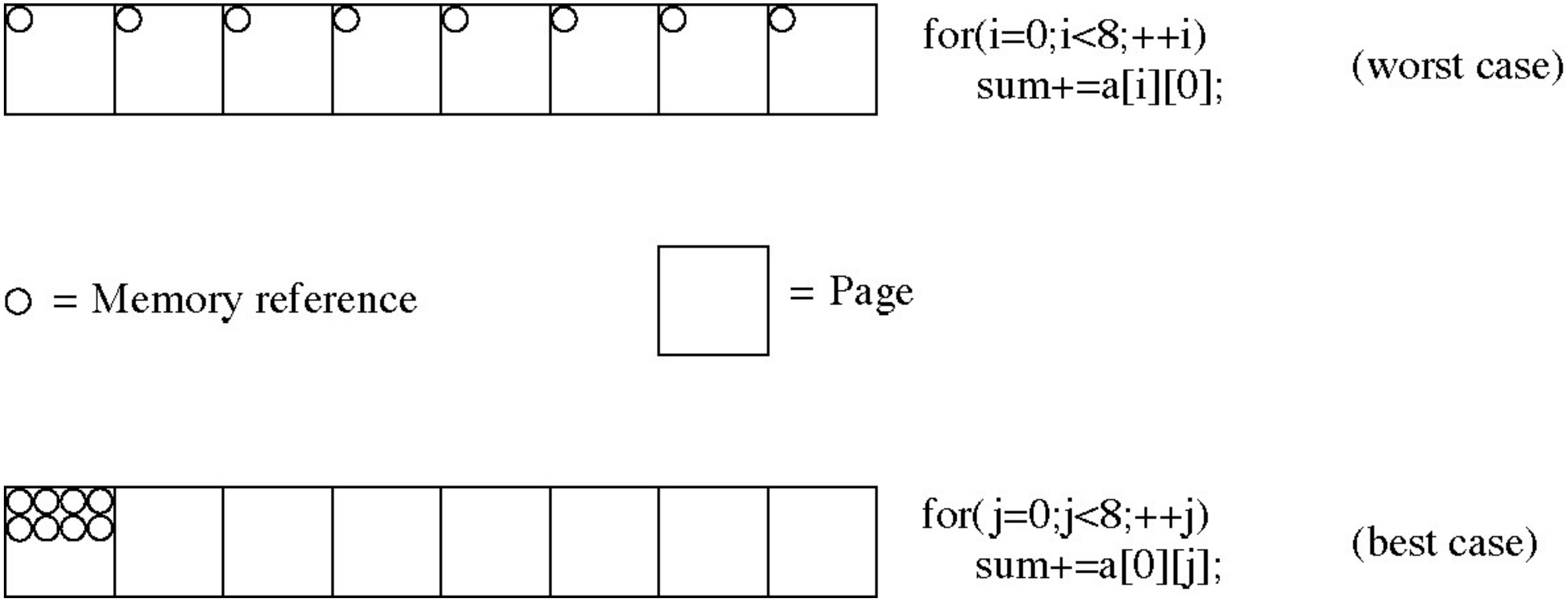
TLB flow chart


Logical address
Page in TLB?
Yes
No
Look up in page table
Store result in TLB
Add offset to page base address
Physical address

Why TLBs work

- Memory access is not random, that is, not all locations in the address space are equally likely to be referenced
- References are localized because
  - sequential code execution
  - loops in code
  - groups of data accessed together
  - data is accessed many times
- This property is called *locality*
- TLB hit rates are 90+% .

Good and bad cases for paging



```
for(i=0;i<8;++i)
    sum+=a[i][0];
```

(worst case)

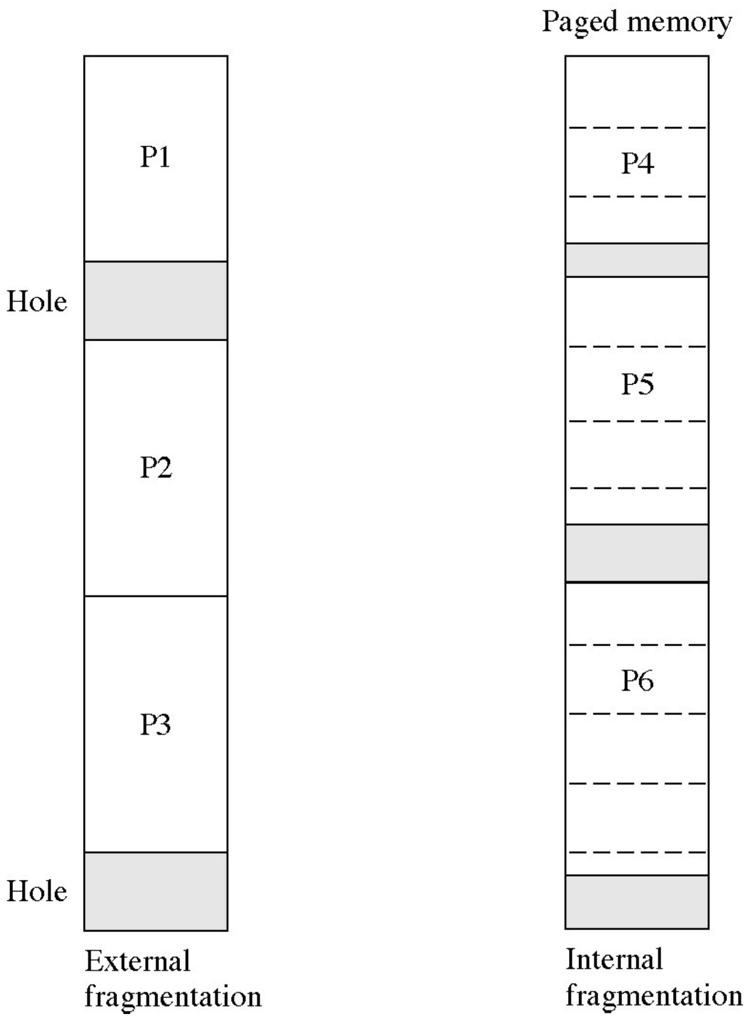
○ = Memory reference

☐ = Page

```
for( j=0;j<8;++j)
    sum+=a[0][j];
```

(best case)

Internal and external fragmentation


Paged memory
P1
Hole
P2
P3
Hole
External fragmentation
P4
P5
P6
Internal fragmentation

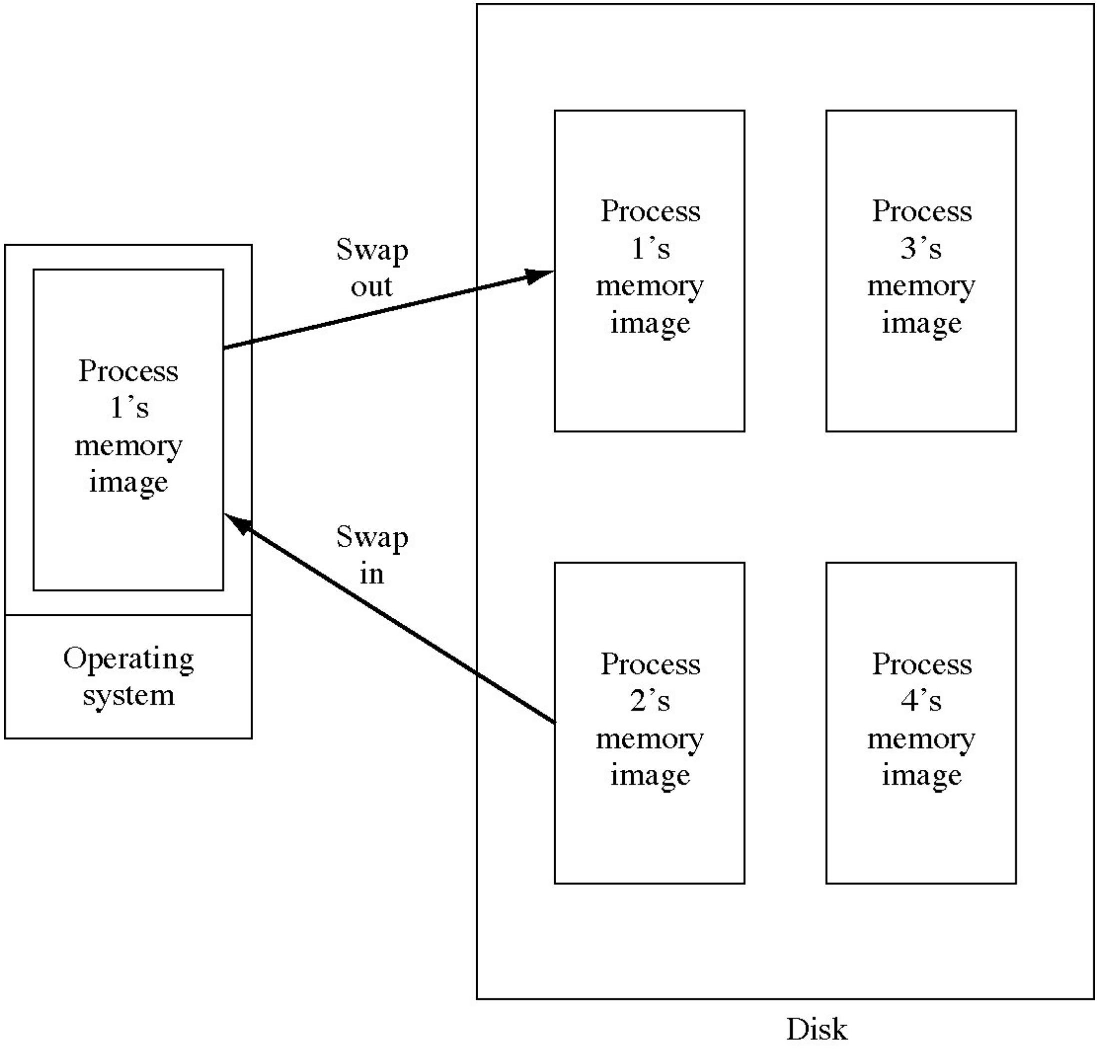
Page and page frame

- Page
  - the information in the page frame
  - can be stored in memory (in a page frame)
  - can be stored on disk
  - multiple copies are possible
- Page frame
  - the physical memory that holds a page
  - a resource to be allocated

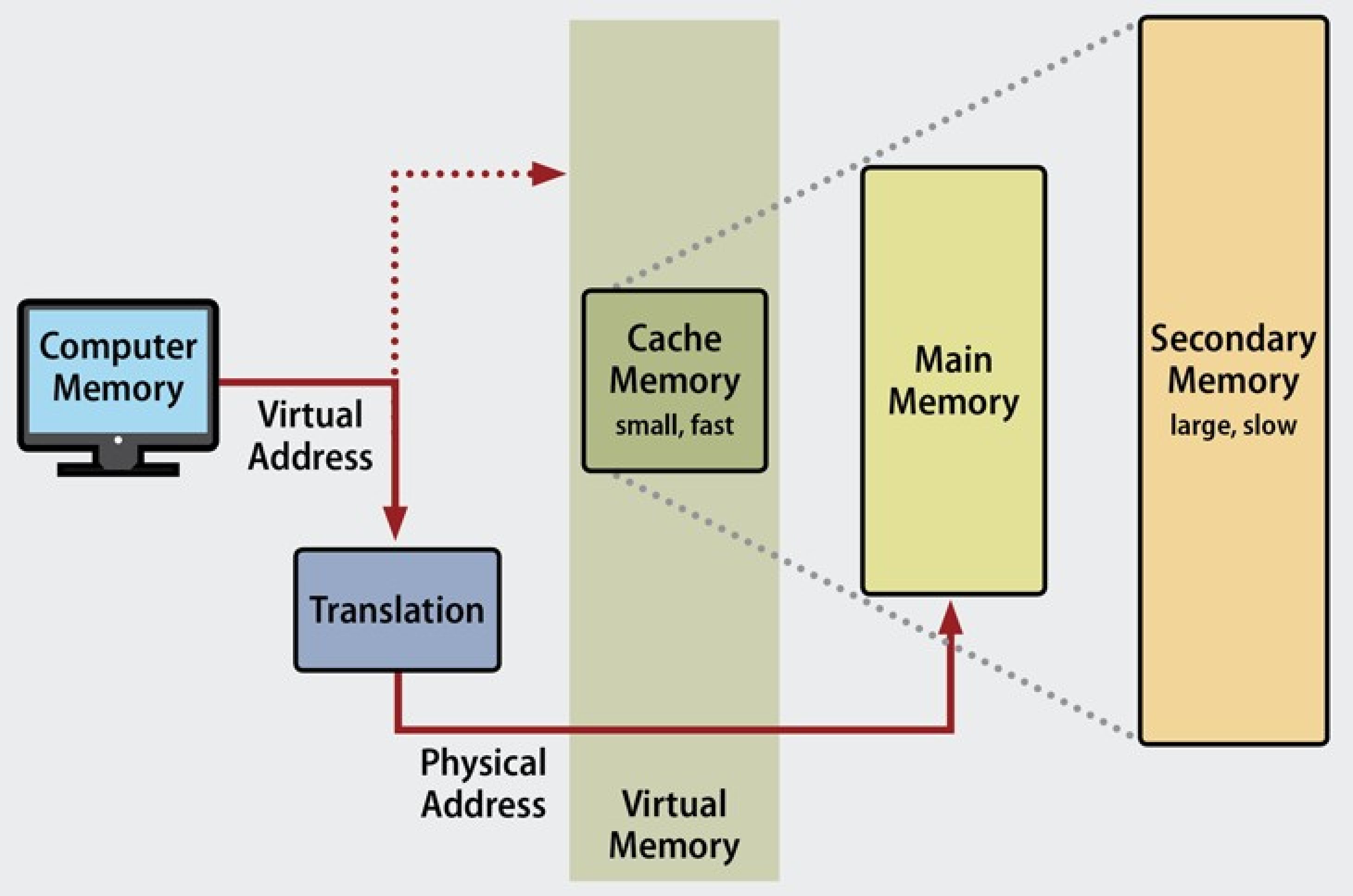
Page table protection

- Three bits control: read, write, execute
- Possible protection modes:
  - 000: page cannot be accessed at all
  - 001: page is read only
  - 010: page is write only
  - 100: page is execute only
  - 011: page can be read or written
  - 101: page can be read as data or executed
  - 110: write or execute, unlikely to be used
  - 111: any access is allowed

Swapping


Process
1's
memory
image
Operating
system
Swap
out
Swap
in
Process
1's
memory
image
Process
3's
memory
image
Process
2's
memory
image
Process
4's
memory
image
Disk

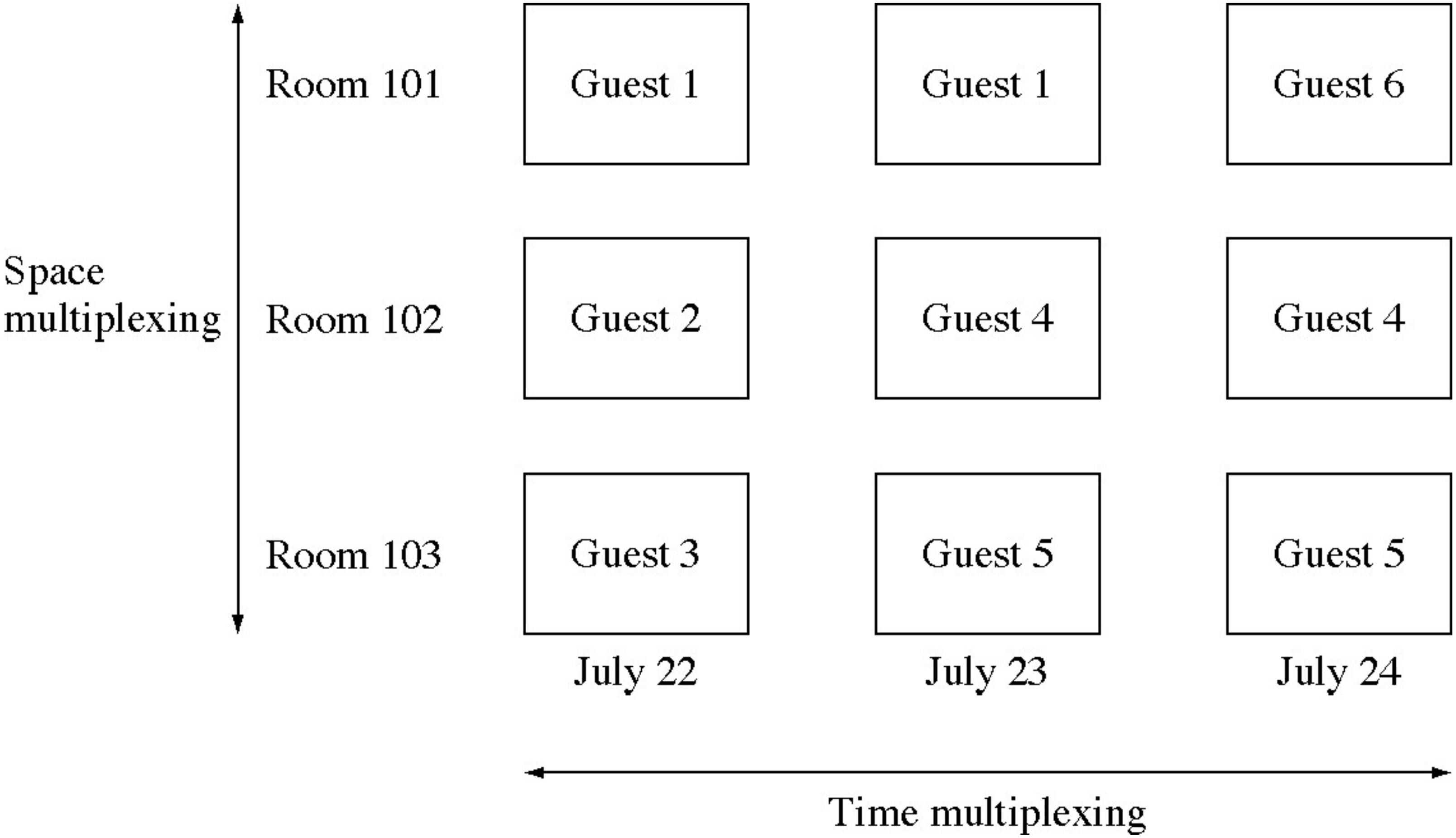
Virtual memory


Computer
Memory
Virtual
Address
Translation
Physical
Address
Cache
Memory
small, fast
Virtual
Memory
Main
Memory
Secondary
Memory
large, slow

Time and space multiplexing

- The processor is a time resource
  - It can only be time multiplexed
- Memory is a space resource
  - We have looked at space multiplexing of memory
  - Now we will look at time multiplexing of memory

Time and space multiplexing of hotel rooms



	July 22	July 23	July 24
Room 101	Guest 1	Guest 1	Guest 6
Room 102	Guest 2	Guest 4	Guest 4
Room 103	Guest 3	Guest 5	Guest 5

Space multiplexing
Time multiplexing

Implementation of virtual memory

- Virtual memory allows
  - time multiplexing of memory
  - users to see a larger (virtual) address space than the physical address space
  - the operating system to split up a process in physical memory
- Implementation requires extensive hardware assistance and a lot of OS code and time
  - but it is worth it

Virtual memory software

- The virtual memory (a.k.a. paging) system in the OS must respond to four events
  - process creation
  - process exit
  - process dispatch
  - page fault

Process creation actions

- 1. Compute program size (say N pages)
- 2. Allocate N page frames of swap space
- 3. Allocate a page table (in the OSs memory) for N page table entries.
- 4. Initialize the swap area
- 5. Initialize the page table: all pages are marked as not present.
- 6. Record the location in the swap area and of the page table in the process descriptor

Process exit actions

- 1. Free the memory used by the page table
- 2. Free the disk space in the swap area
- 3. Free the page frames in process was using

Process dispatch actions

- 1. Invalidate the TLB (since we are changing address spaces)
- 2. Load the hardware page table base register with the address of the page table for this process

Page fault actions

- 1. Find the faulting page (say page K)
- 2. Find an empty page frame. This will involve replacing a page.
- 3. Read in page K to this page frame
- 4. Fix up the page table entry for page K. Mark it present and set the base address.
- 5. Restart the process with the instruction that caused the page fault

Locality

- Programs do not access their address space uniformly
  - they access the same location over and over
- *Spatial locality*: processes tend to access location near to location they just accessed
  - because of sequential program execution
  - because data for a function is grouped together
- *Temporal locality*: processes tend to access data over and over again
  - because of program loops
  - because data is processed over and over again

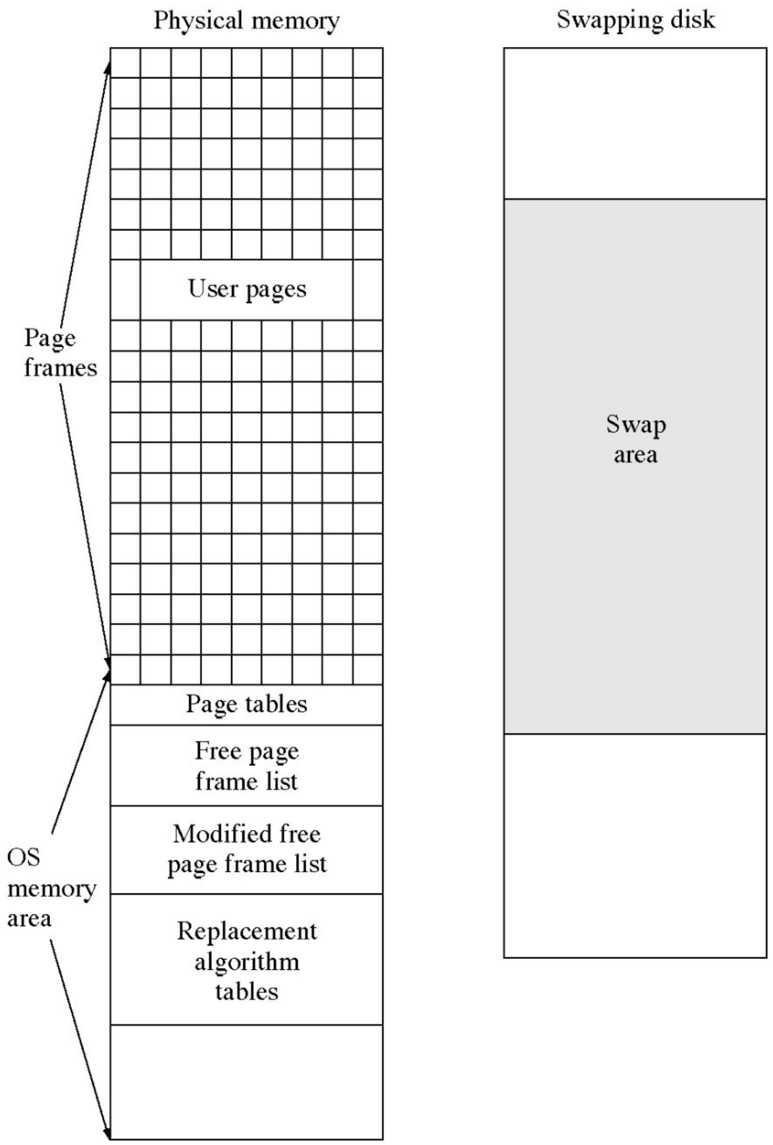
Design technique: Locality

- Locality is almost always present
  - and often we can optimize a design by taking advantage of it.
- Caching is a common name for systems that take advantage of locality to optimize operations

Practicality of paging

- Paging only works because of locality
  - at any one point in time programs don’t need most of their pages
- Page fault rates must be very, very low for paging to be practical
  - like one page fault per 100,000 or more memory references

VM data structures


Physical memory
Swapping disk
User pages
Page
frames
Swap
area
Page tables
Free page
frame list
Modified free
page frame list
OS
memory
area
Replacement
algorithm
tables

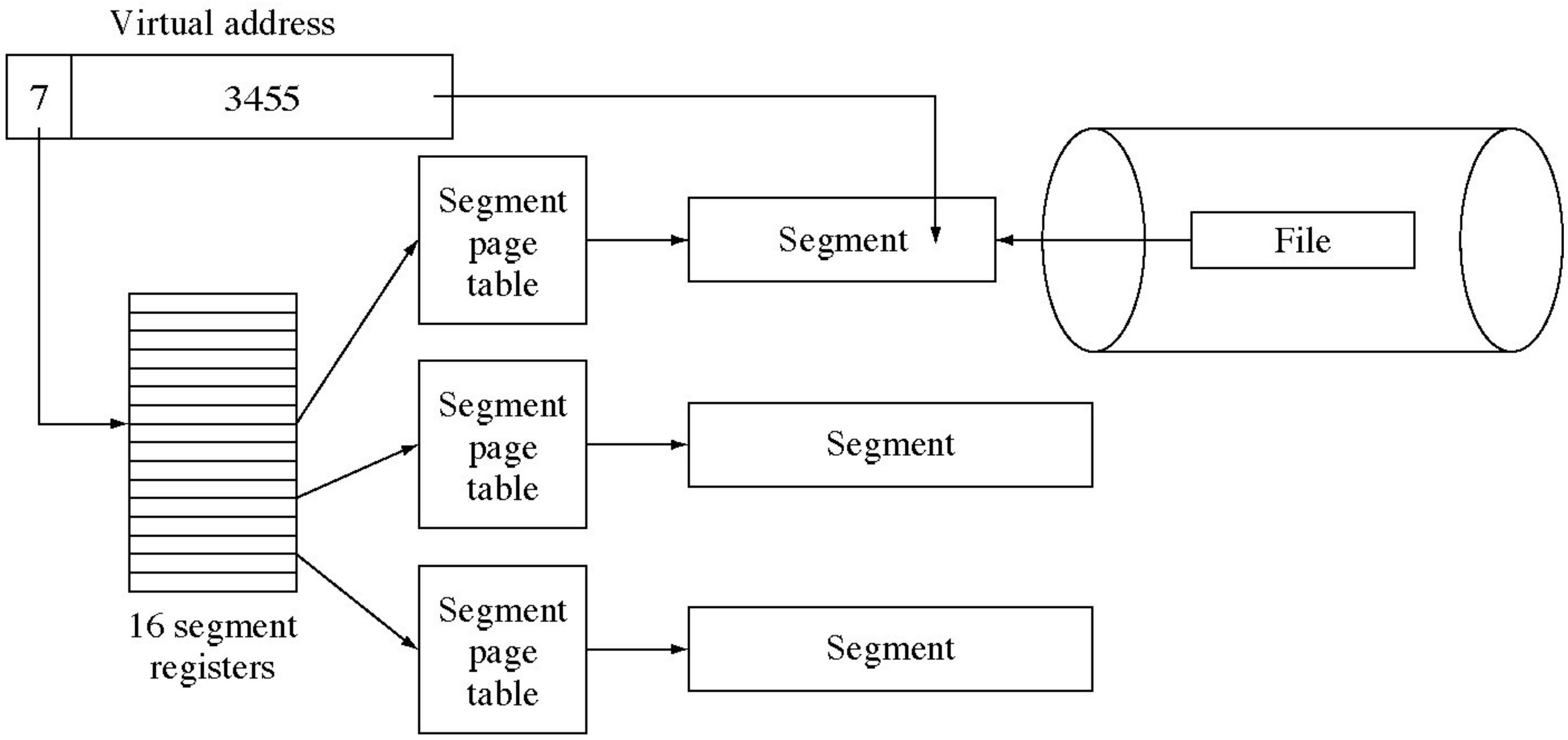
Reactive and proactive user interfaces

- Graphical user interfaces are generally reactive, they wait for user actions
  - this is a good model and puts the user in control which is good psychologically
- Agents are a new user interface concept
  - agents are proactive
  - they go out and look for useful things to do

Design technique: Polling, software interrupts and hooks

- How can a process know when an event occurs? There are two approaches
- *polling*: it can check periodically
- *interrupts*: it can ask another process to interrupt it when the event occurs
  - the other process is usually the one that causes or handles the event
  - so it is not much trouble for it to inform the waiting process that the event has ocurred.

Virtual memory in the IBM 801


Virtual address
7
3455
Segment page table
Segment
File
Segment page table
Segment
Segment page table
Segment
16 segment registers

File mapping

- *File mapping* is the mapping of a file on disk into the virtual address space of a process.
- File I/O then consists of reading and writing words in the virtual address space
  - no system calls are required for read and write
- This is also called a *memory-mapped file*.
- The I/O system and the paging system both move data between disk and memory
  - so it makes sense to combine them

THANK YOU

BITS Pilani
Pilani Campus

BITS Pilani
Pilani Campus

COMPUTING AND DESIGN

SESSION 9

Course design by - Dr. Lucy J. Gudino &
Dr Chandrasekhar
Faculty – Dr Thangakumar J
WILP & Department of CS & IS

BITS Pilani
Pilani Campus

Operating systems – I/O Abstraction

Last Session

List of Topic Title	Text/Ref Book/external resource
• Fragmentation and compaction	T1 (11.1)
• Page table location : Hardware register Vs. Main memory Vs Cache	T1(11.2.4, 11.2.5, 11.2.6)
• Implementing Virtual memory	T1(11.7)

I/O Abstraction

Today's topic

- **I/O System Hardware**
  - Device Controllers
  - Types of devices : Terminal Devices, Communication Devices and Disk Devices
- **Data transfer schemes**
  - Programmed I/O, Interrupt Driven I/O, DMA

INPUT OUTPUT – STORY LINE

1

POLLING
  - CHECK
  - INITIATE
  - COMPLETE

3

DIRECT MEMORY ACCESS

- INPUT OUTPUT IN BLOCKS
- WHICH DEVICE, HOW MANY BLOCKS, MEMORY LOCATION.
- CPU “interrupted” to check for errors; assigns next program

2

- INTERRUPTIONS
  - SAVE CONTEXT IN MEMORY
  - INTERRUPT SERVICE ROUTINE
  - LOOKUP TO EXECUTE INTERRUPT SERVICE ROUTINE
  - RESTORE CONTEXT AND RESUME OPERTIONS

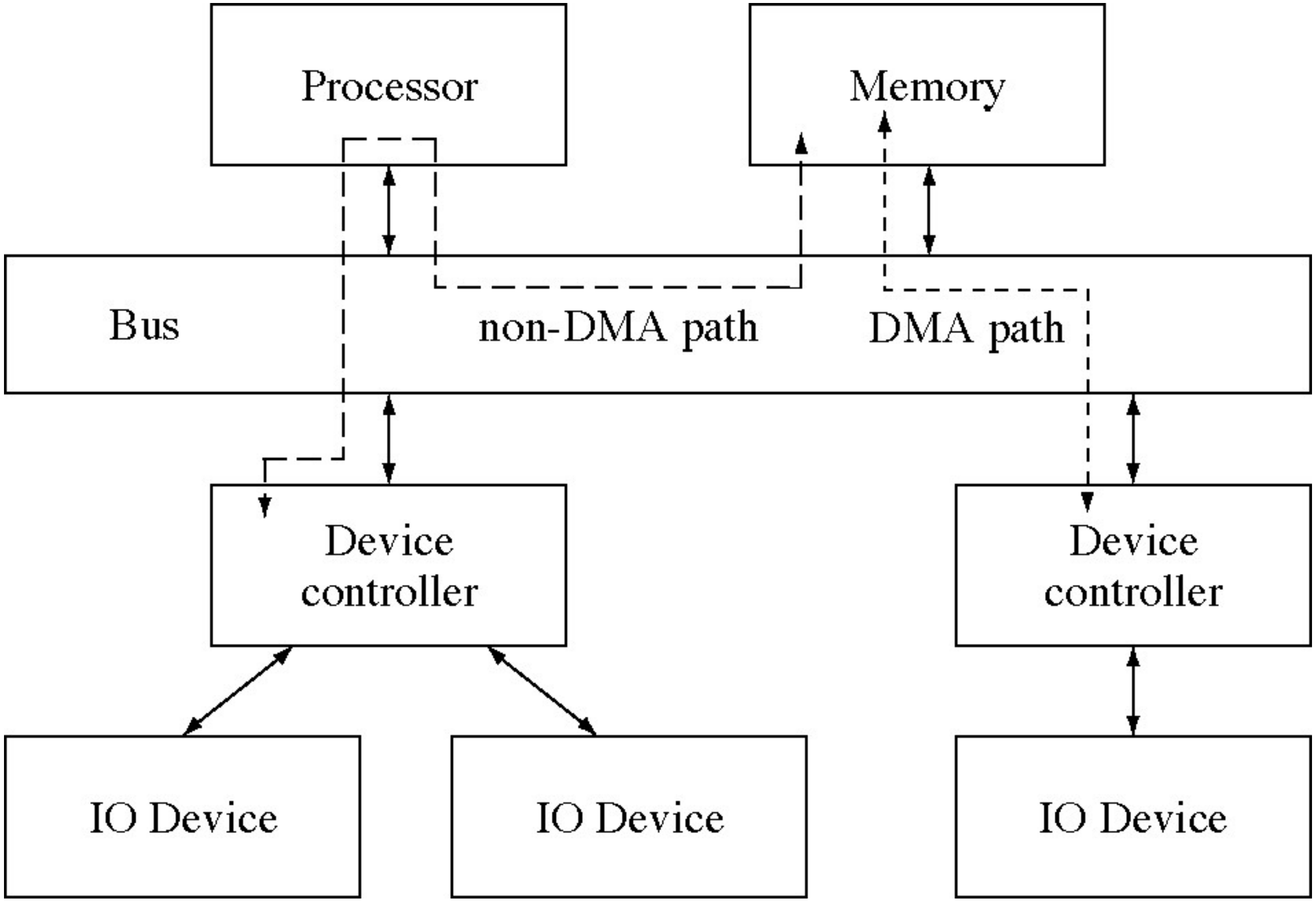
4

- INPUT OUTPUT PROCESSORS
  - IOP CAN MANAGE MANY DEVICES
  - IOP CAN MANAGE HIGH LEVEL IOP
  - MORE SOPHISTICATED THAN DMA

DEVICES AND CONTROLLERS

- ***Input device*: **Generates data for a computer to read.
- ***Output device*: **Accepts data from computer.
- ***Device controller*: **an intermediate electronic device that interfaces between the computer system bus and one or more devices
- ***I/O processor or channel*: **a programmable device controller

Device Controller and I/O Devices


Processor
Memory
Bus
non-DMA path
DMA path
Device controller
Device controller
IO Device
IO Device
IO Device

- An interface Unit

- Communicates with both I/O devices and computer system (over bus)

- Figure shows the connection between the I/O devices, the device controller and the system bus.

- **I/O Device** – transfer of data functionality; **I/O Controller** – interface to the computer functionality.

- **I/O Processor (Channels)** – Device controller know how to execute programs of more than one instruction.

- A controller with Direct Memory Access (DMA) can directly read and write the memory in the system. This can transfer data faster than non-DMA controller.

TERMINAL DEVICES

- **Terminal** – a device that allows a user to communicate with a computer. *Screen, Keyboard*.
- **Serial Port** – A connection which allows communication of a stream of bits.
- **ASCII code** - display-a-character; E.g., display command 65 represents A.
- Fixed number of lines and columns – 24*80 (1960 bytes)
- The screen is refreshed from 30 to 60 times a second.
- **Screen Refreshing** - A beam of electrons at a special chemical (Phosphor) on the inside of the screen that glows for a short time after it has been hit by the electron beam

- **Display** – Glass tube (Cathode Ray Tube or CRT)

- **Scan Line** – a line from the upper left corner and moves to the right horizontally. Fixed number of dot positions are available along the horizontal line.

- **Pixel** – dot position

- **Horizontal Retrace** – beam turned off (Pixel goes to right end of horizontal line) and moved back to the left end of the screen (previous line starts). Continues till it reaches a lowest right corner.

- **Vertical Retrace** – moves back up to the upper left corner (beam is off)

- **Character Generator Memory** – Converts an ASCII Characters from screen memory into an array of 8 by 14 dots that can be written on the screen.
- Shape of the character is changed by changing the content of Character Generator Memory (ROM chip or writable screen generator memory).
- ASCII codes (7 bit long), screen memory uses (8 bit) bytes. 128 ASCII characters are not used.
- The keyboard generates character codes for the keys that are typed, and these character codes are sent on to the device controller.

Mouse Devices

- Connected either with Keyboard or directly to the computer by its own serial port. So mouse provides two types of inputs.
  - keyboard events
    - key down
    - key up
  - mouse event
    - mouse button down
    - mouse button up
    - mouse movement
- It is a relative device reports the change in position in the two direction.
- Its button reports when it is pressed or released.
- Coded as multibyte records.

Color and Color Maps

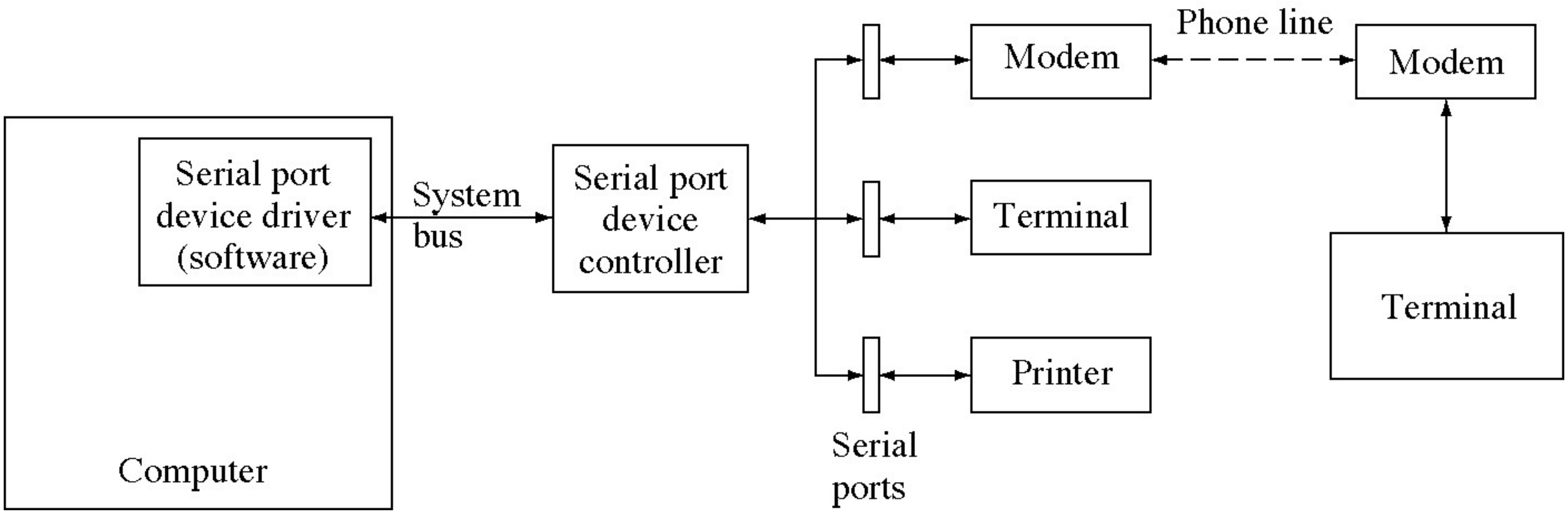
- Usually specified with 24 bits.
- **Color Map** – a set of 256 24-bit registers which map 8-bit color identifiers into 24-bit color specifications.
- Managed by window manager.

Modems

- **Modulation** – the process of converting a bits to sound
- **Demodulation** – reverse process
- **Modem (Modulator – demodulator)** – the hardware device that knows how to do this.
- Modem devices are always paired because signals are modulated on one end and demodulated on the other end.

COMMUNICATION DEVICES – Serial Port

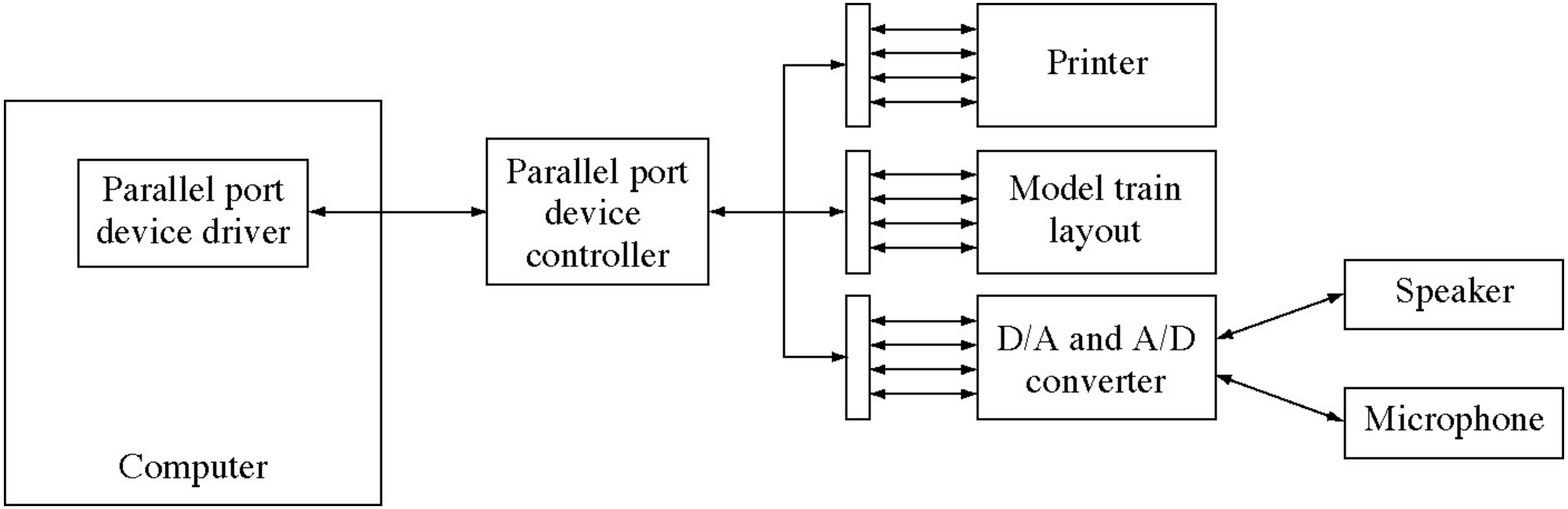
- **A Serial Port Controller** – Connects to a system bus and provides one or more serial ports for devices.
- **Example:** a printer, a mouse


Phone line
Modem
Modem
Serial port device driver (software)
System bus
Serial port device controller
Terminal
Terminal
Printer
Computer
Serial ports

Parallel Ports

- It has 8 or 16 lines for communicating information and so can transfer 8 or 16 bits at a time.

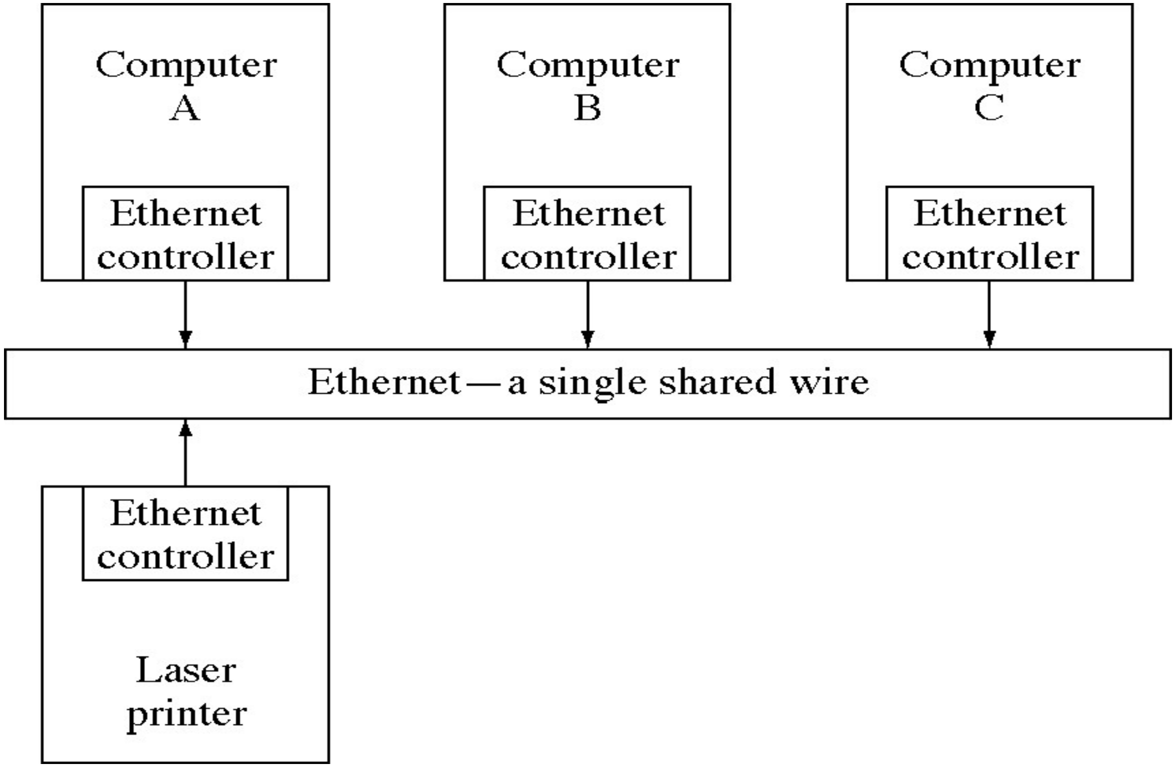
- A **parallel port controller** – Connected to the system bus and provides one or more parallel ports.


Parallel port device driver
Computer
Parallel port device controller
Printer
Model train layout
D/A and A/D converter
Speaker
Microphone

Ethernet Devices

- **A Ethernet** – technology that allows to connect a number of computers together.

- **Packets** – packages of bits that are transferred in a unit.


Computer A
Ethernet controller
Computer B
Ethernet controller
Computer C
Ethernet controller
Ethernet—a single shared wire
Ethernet controller
Laser printer

Magnetic Disk Drive

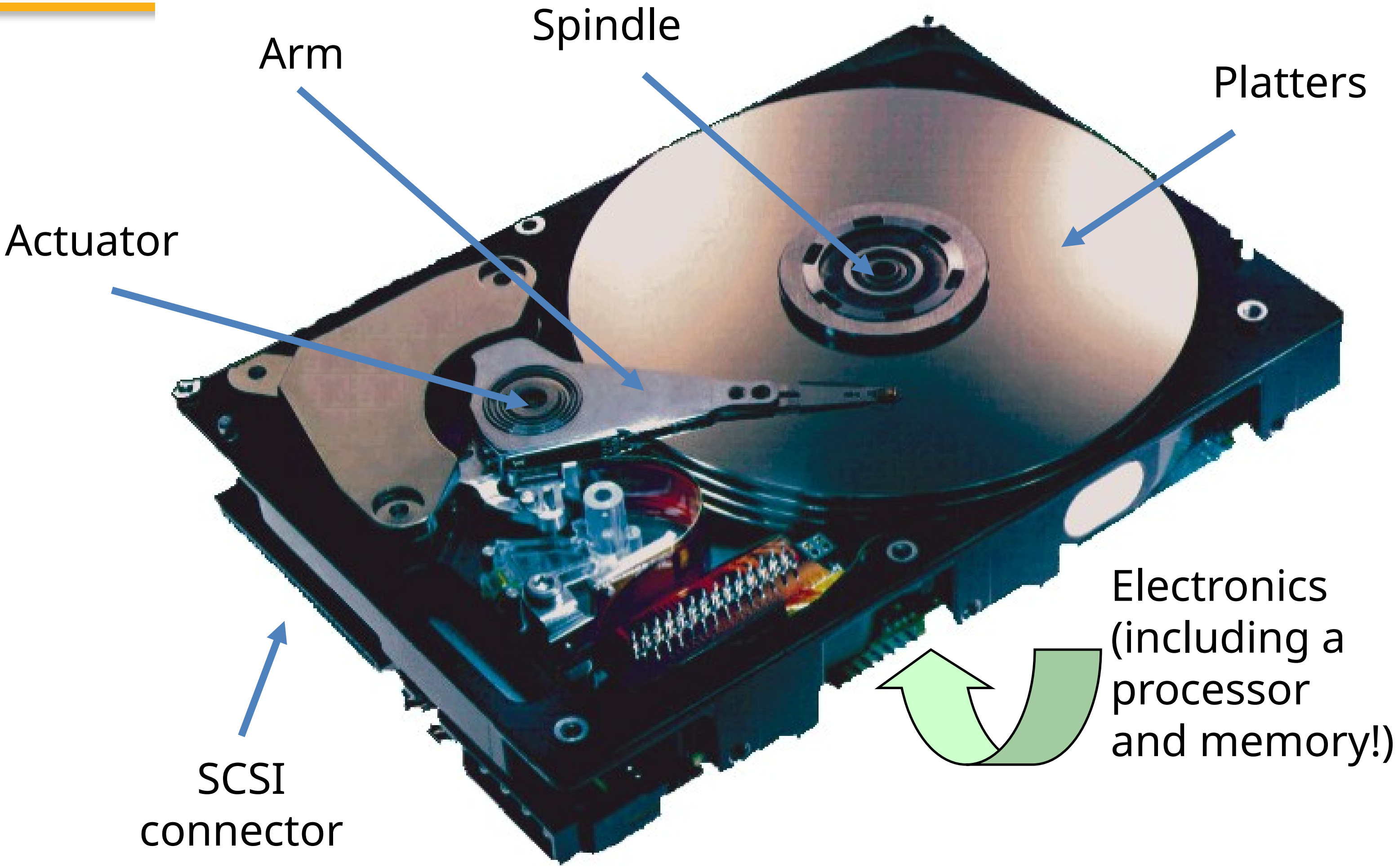
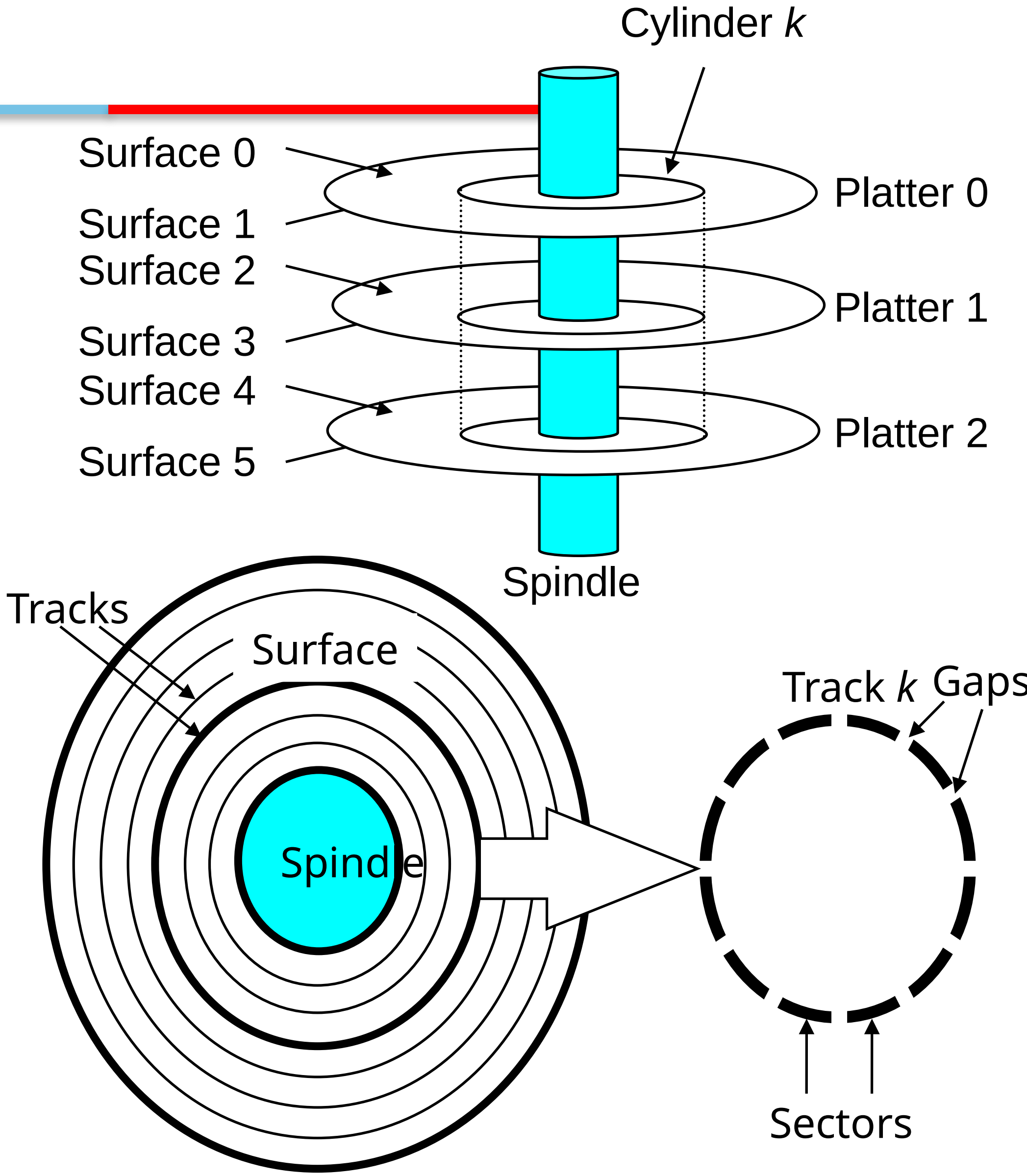

Spindle
Arm
Platters
Actuator
Electronics (including a processor and memory!)
SCSI connector

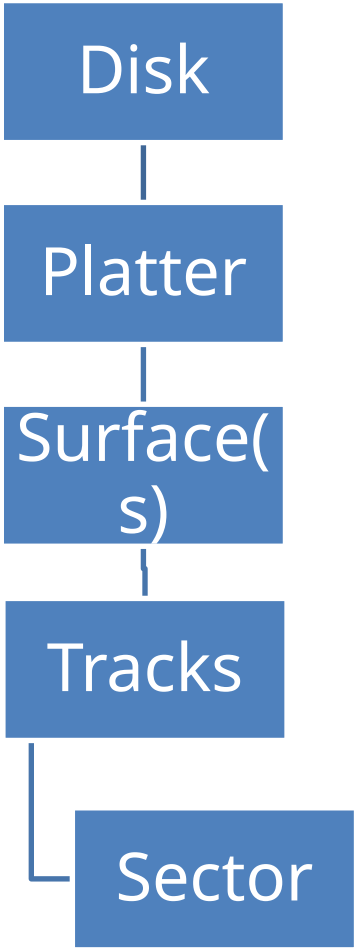
Image courtesy of Seagate Technology

Disk Geometry

- Disks consist of platters, each with two surfaces.
- Each surface consists of concentric rings called tracks
- Aligned tracks form a cylinder
- Each track consists of sectors separated by gaps.


Cylinder *k*
Surface 0
Surface 1
Surface 2
Surface 3
Surface 4
Surface 5
Platter 0
Platter 1
Platter 2
Spindle
Tracks
Surface
Spindle
Track *k*
Gaps
Sectors

Disk Geometry (contd)



Disk

Platter

Surface(s)

Tracks

Sector

Disk Capacity

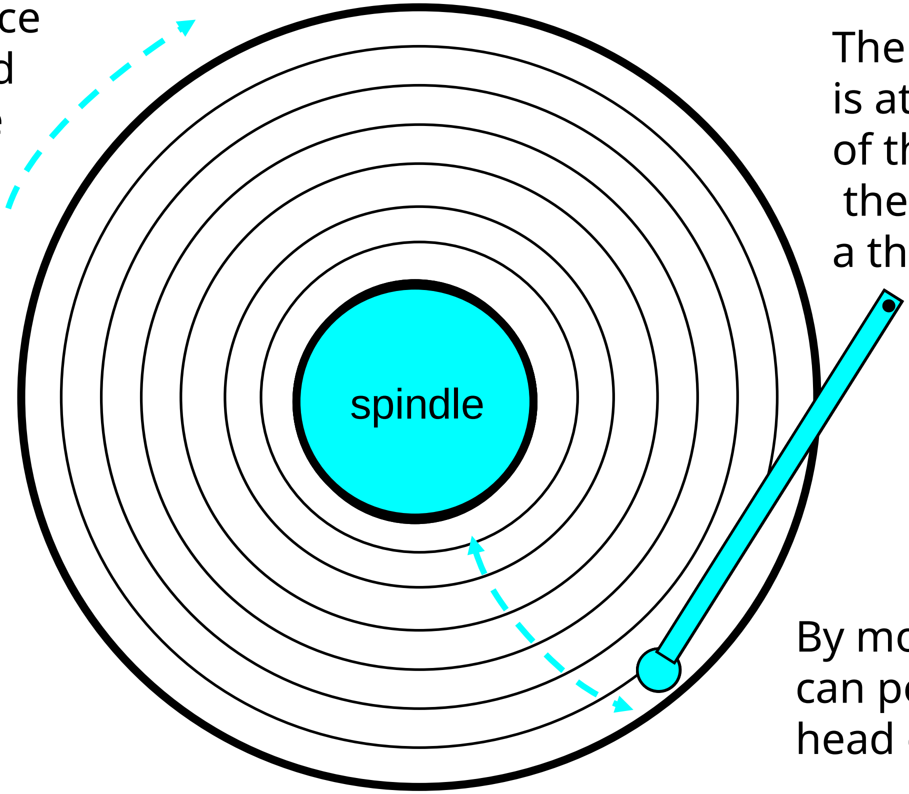
- Capacity: maximum number of bits that can be stored.
  - Vendors express capacity in units of gigabytes (GB /TB), where 1 GB = 2^{30} Bytes, 1 TB = 2^{40} Bytes,
- Capacity is determined by these technology factors:
  - Recording density (bits/in): number of bits that can be squeezed into a 1 inch segment of a track.
  - Track density (tracks/in): number of tracks that can be squeezed into a 1 inch radial segment.
  - Areal density (bits/in2): product of recording and track density.

Computing Disk Capacity

- Capacity = (# bytes/sector) x (avg. # sectors/track) x (# tracks/surface) x (# surfaces/platter) x (# platters/disk)
- Example:
  - 512 bytes/sector
  - 300 sectors/track (on average)
  - 20,000 tracks/surface
  - 2 surfaces/platter
  - 5 platters/disk

- Capacity = 512 x 300 x 20000 x 2 x 5
  = 30,720,000,000
  = 28.61 GB

Disk Operation (Single-Platter View)

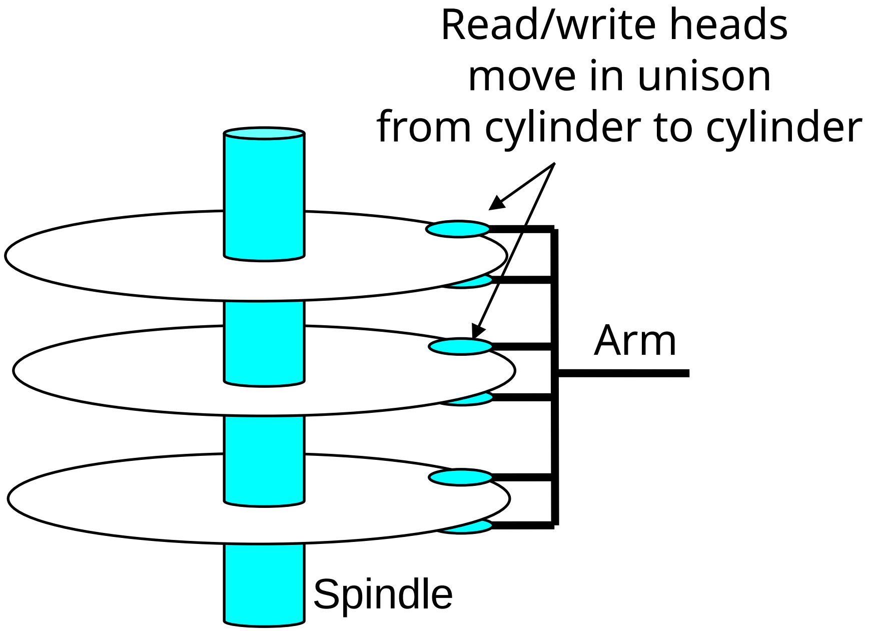


The disk surface spins at a fixed rotational rate

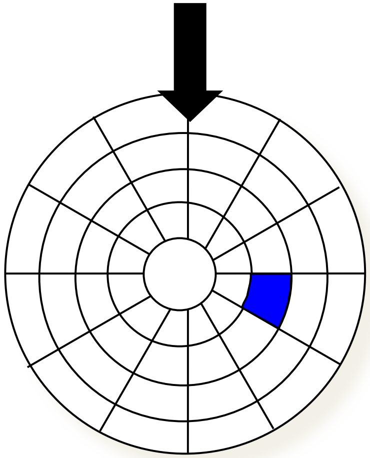
The read/write *head* is attached to the end of the *arm* and flies over the disk surface on a thin cushion of air.

By moving radially, the arm can position the read/write head over any track.

Disk Operation (Multi-Platter View)

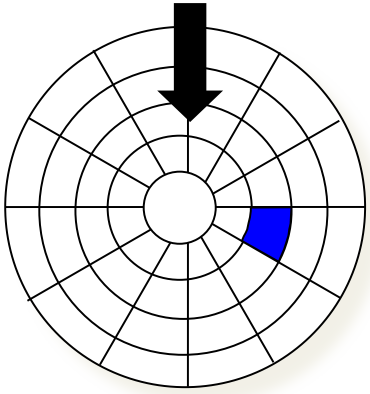

Read/write heads
move in unison
from cylinder to cylinder
Arm
Spindle

Disk Access



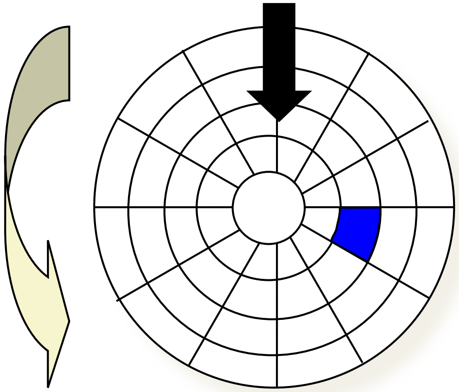
Need to access a sector colored in blue

Disk Access



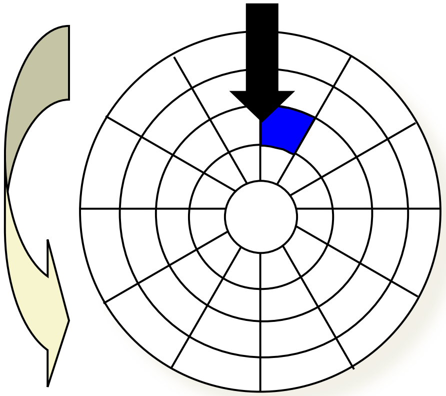
Head in position above a track

Disk Access



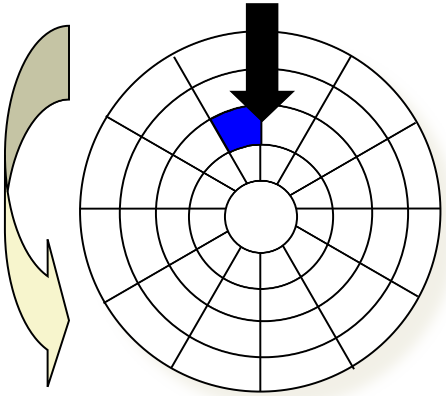
Rotate the platter in counter-clockwise direction

Disk Access – Read



About to read blue sector

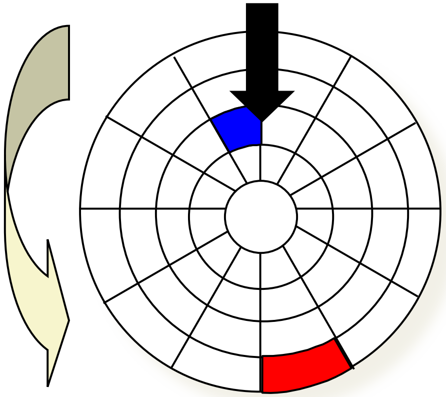
Disk Access – Read



After BLUE read

After reading blue sector

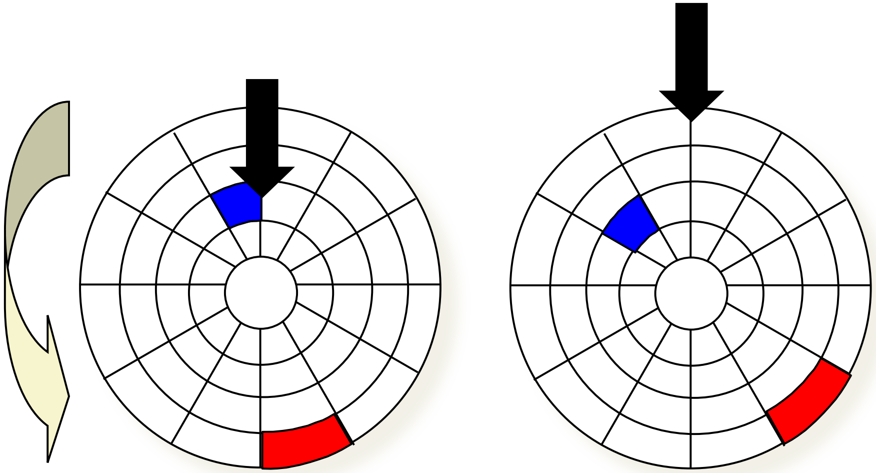
Disk Access – Read



After BLUE read

Red request scheduled next

Disk Access – Seek

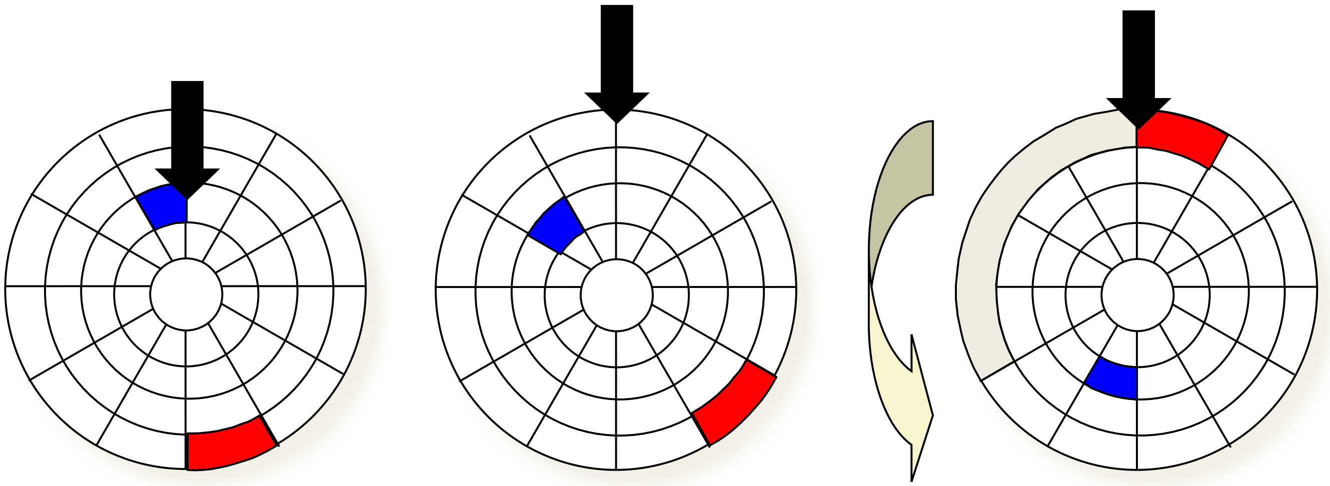


After BLUE read

Seek for RED

Seek to red's track

Disk Access – Rotational Latency



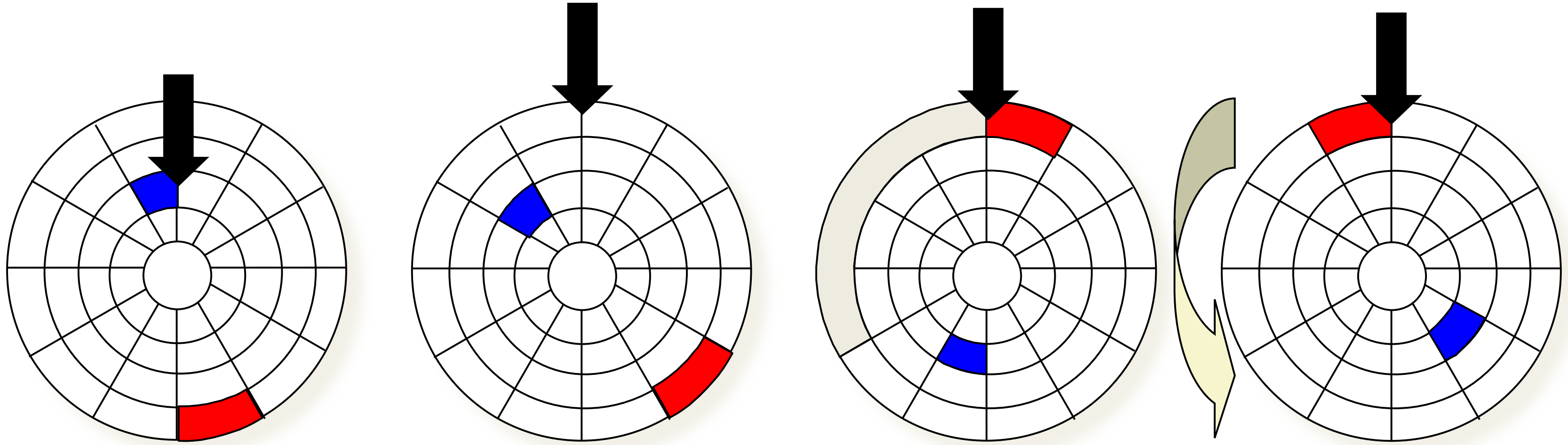
After BLUE read

Seek for RED

Rotational latency

Wait for red sector to rotate around

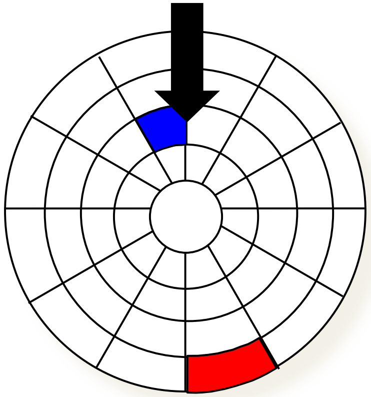
Disk Access – Read



After BLUE read Seek for RED Rotational latency After RED read

Complete read of red

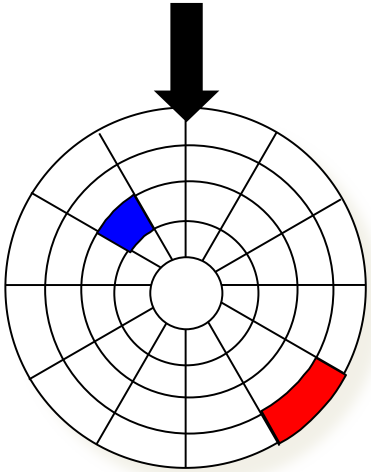
Disk Access – Access Time Components



After BLUE read

↑

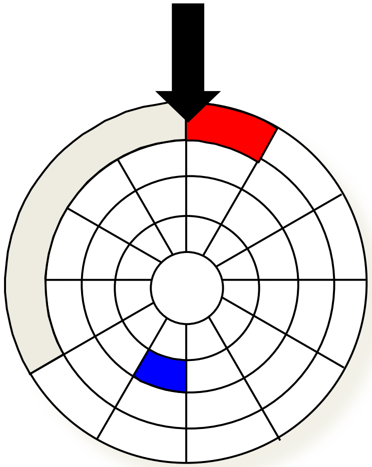
Data transfer



Seek for RED

↑

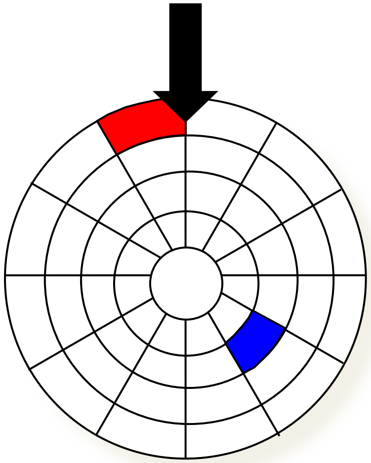
Seek



Rotational latency

↑

Rotational latency



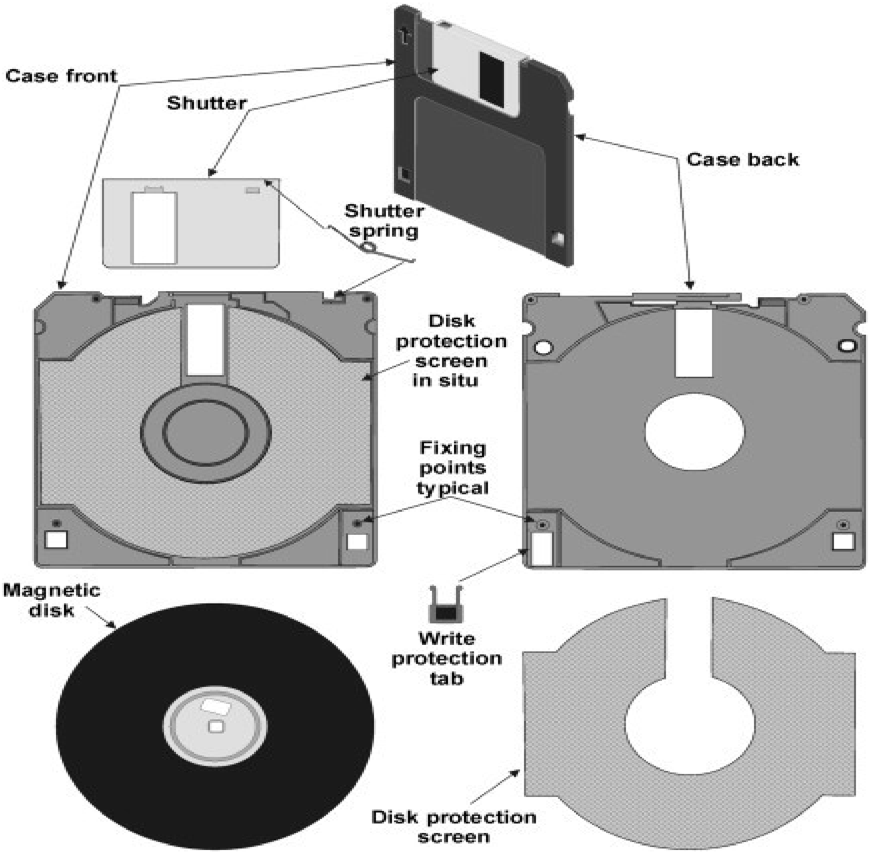
After RED read

↑

Data transfer

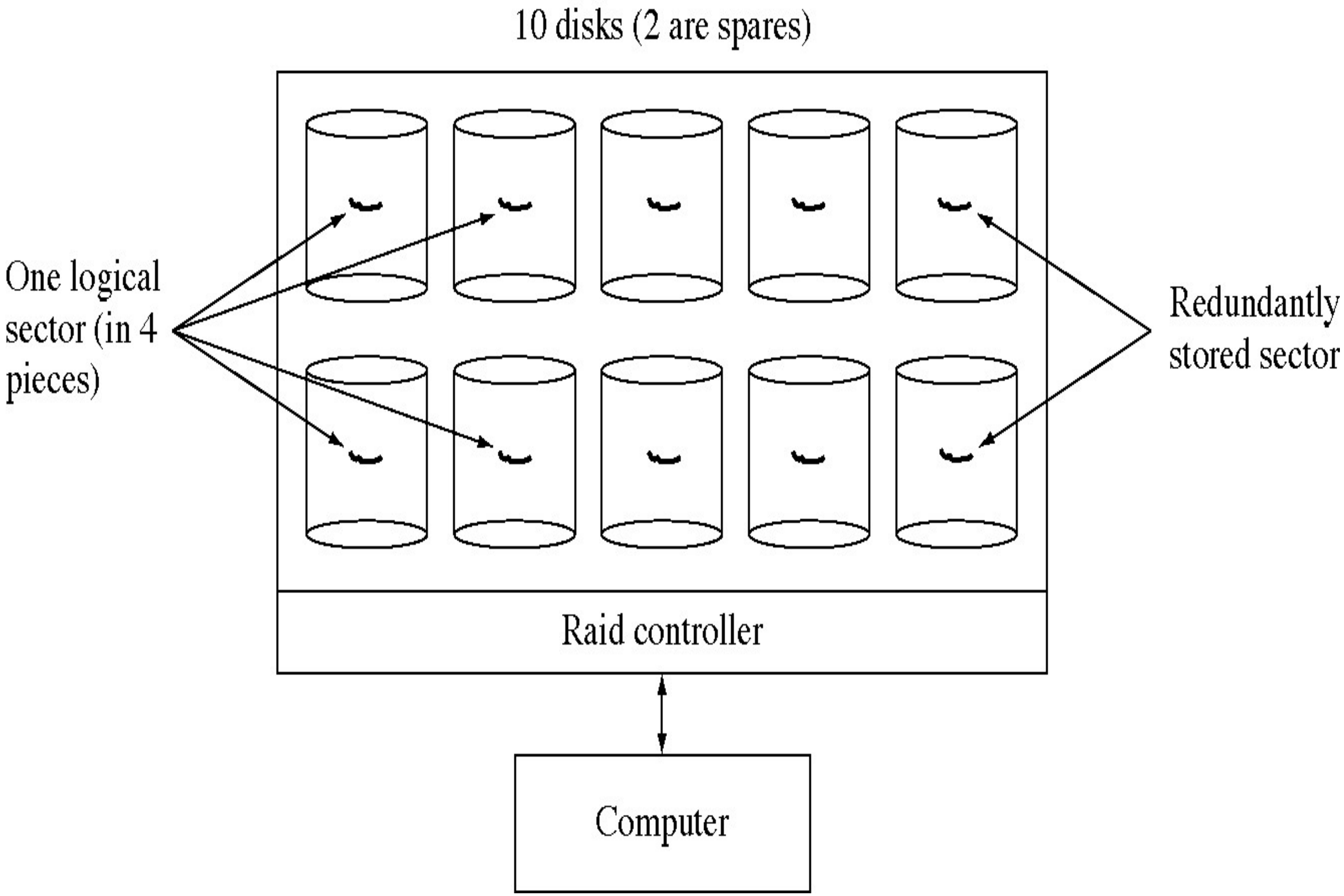
Floppy Disks

- Rigid disk, maximum speed and capacity but expensive and media are not removable.


Case front
Shutter
Case back
Shutter spring
Disk protection screen in situ
Fixing points typical
Magnetic disk
Write protection tab
Disk protection screen

RAID Devices

- Disk can only spin fast
  - to increase speed, parallelism is used
- ***RAID*: redundant array of inexpensive disks**
  - **redundant**: RAID can be used in increase reliability through redundancy
  - **array**: RAID uses disks in parallel
  - **inexpensive**: RAID uses disks that are manufactured in the highest volume and are therefor have the best performance/cost ratio



10 disks (2 are spares)

One logical sector (in 4 pieces)

Redundantly stored sector

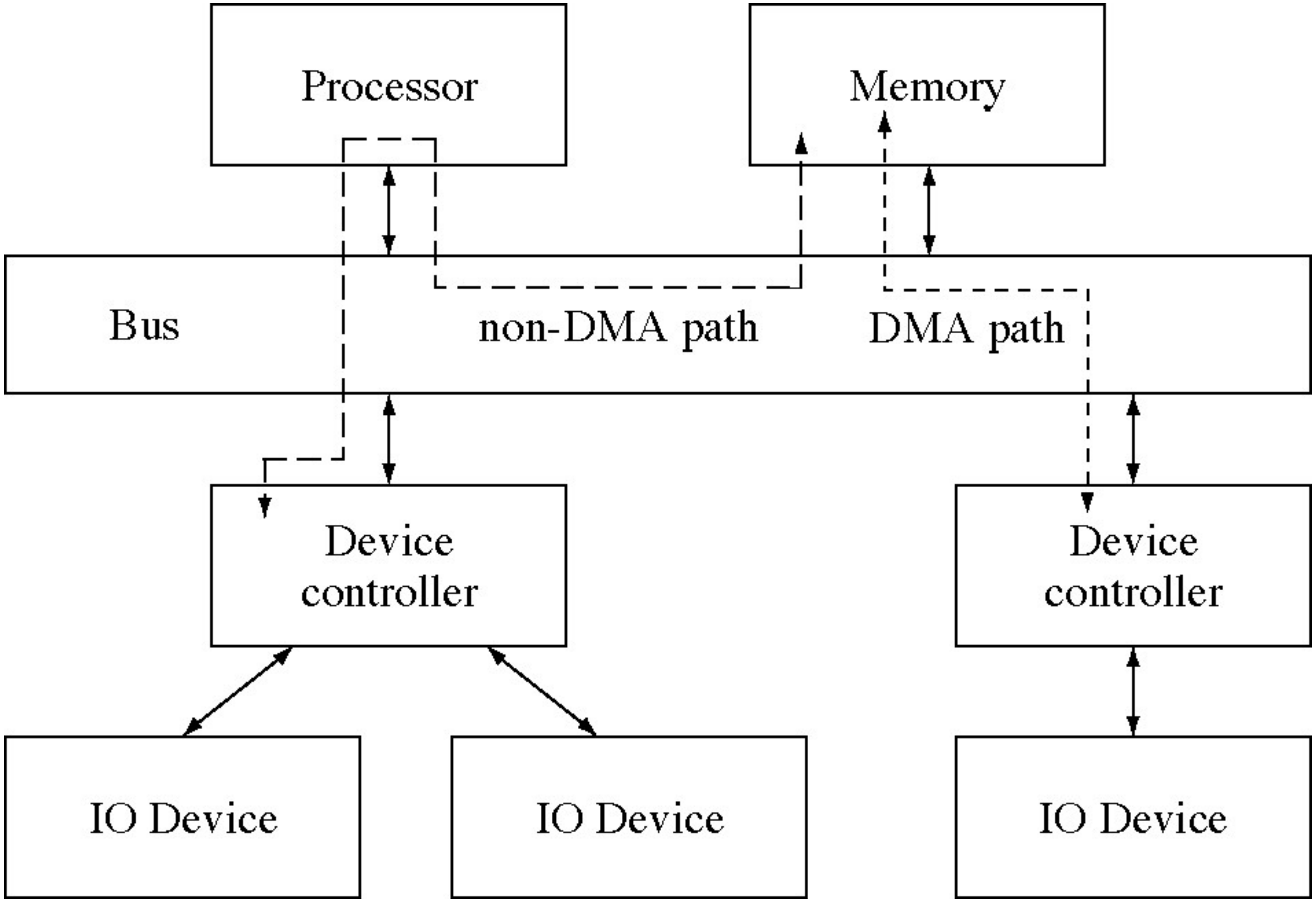
Raid controller

Computer

I/O Methods

- Programmed
- Interrupt driven
- Direct Memory Access (DMA)

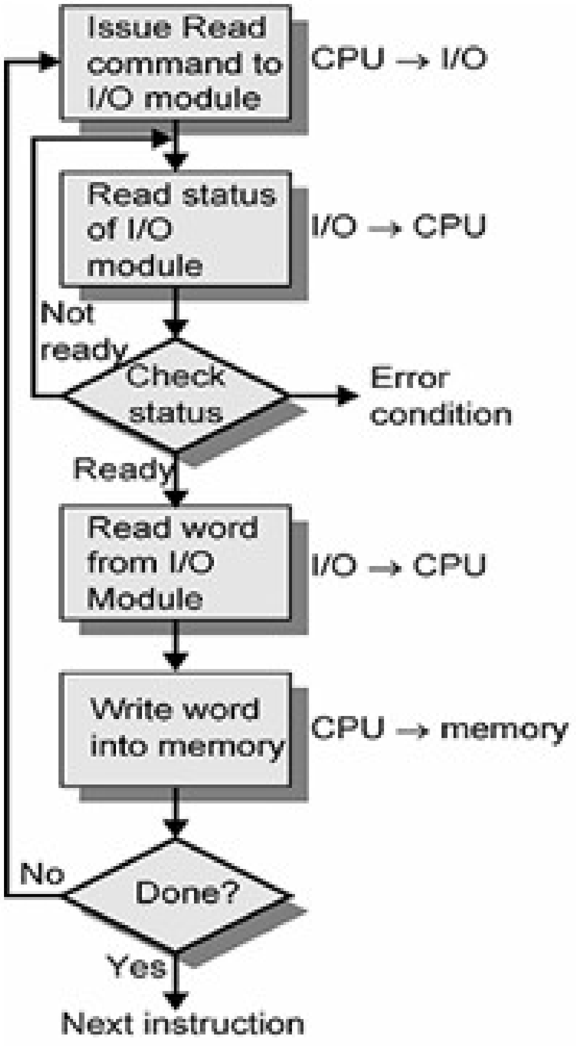
Device Controller and I/O Devices



Processor
Memory
Bus
non-DMA path
DMA path
Device controller
Device controller
IO Device
IO Device
IO Device

Programmed I/O

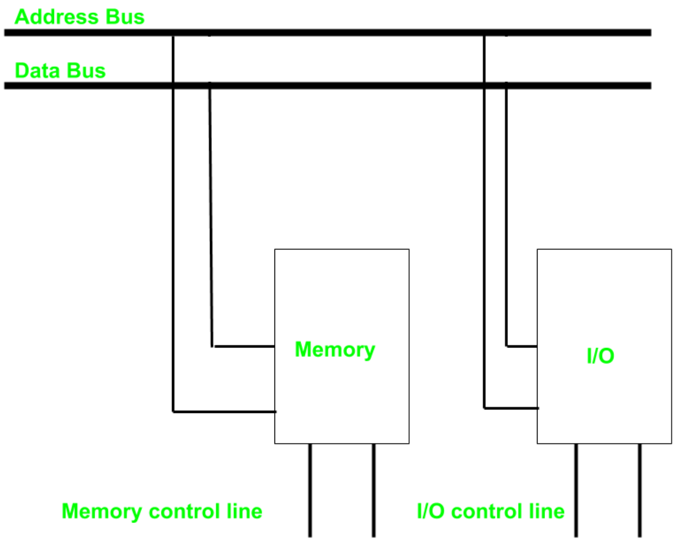
- CPU has direct control over I/O
  - Sensing status
  - Read/write commands
  - Transferring data
- CPU waits for I/O module to complete operation
- Commands
  - Control - telling module what to do
  - Test - check status
  - Read/Write
- Wastes CPU time!!! Why?


Issue Read command to I/O module
CPU → I/O
Read status of I/O module
I/O → CPU
Not ready
Check status
Error condition
Ready
Read word from I/O Module
I/O → CPU
Write word into memory
CPU → memory
No
Done?
Yes
Next instruction

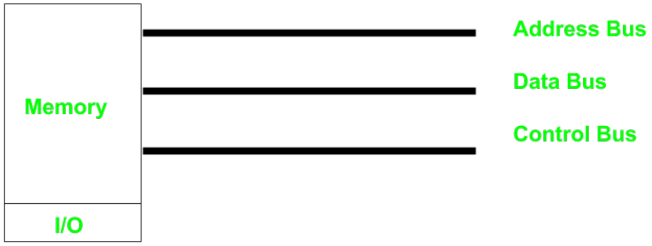
Addressing I/O Devices

- Memory mapped I/O
  - Devices and memory share an address space
  - I/O looks just like memory read/write
  - No special commands for I/O
    - Large selection of memory access commands available
- Isolated I/O
  - Separate address spaces
  - Need I/O or memory select lines
  - Special commands for I/O

Isolated I/O

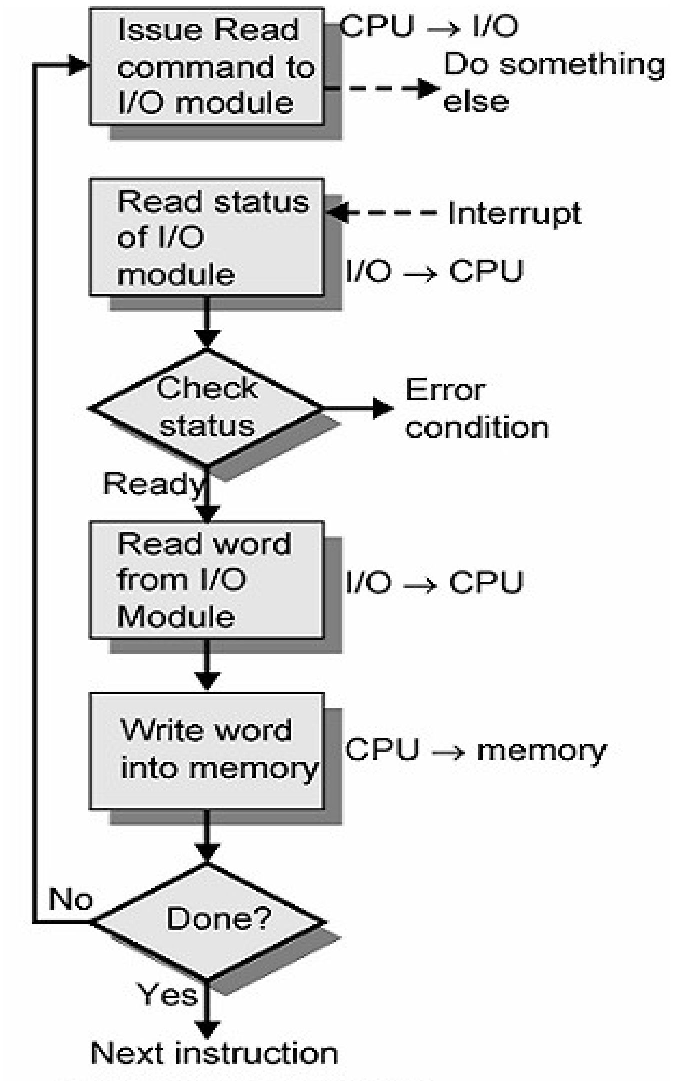

Address Bus
Data Bus
Memory
I/O
Memory control line
I/O control line

Memory mapped I/O


Memory
I/O
Address Bus
Data Bus
Control Bus

Interrupt Driven I/O

- CPU issues read command
- I/O module gets data from peripheral whilst CPU does other work
- I/O module interrupts CPU once data ready
- If interrupted:-
  - Save context (registers values)
  - Process interrupt
    - Fetch data & store


Issue Read command to I/O module
CPU → I/O
Do something else
Read status of I/O module
Interrupt
I/O → CPU
Check status
Error condition
Ready
Read word from I/O Module
I/O → CPU
Write word into memory
CPU → memory
No
Done?
Yes
Next instruction

(b) Interrupt-driven I/O

Device Identification-1

- How does processor determine which device issued the interrupt?
  - Multiple Interrupt Lines
    - Impractical to provide one line to each I/O device as number of devices increase
  - Software Poll
    - Processor branches to ISR (Interrupt service routine)
    - ISR poll each I/O module to identify which module caused the interrupt
    - Time consuming process

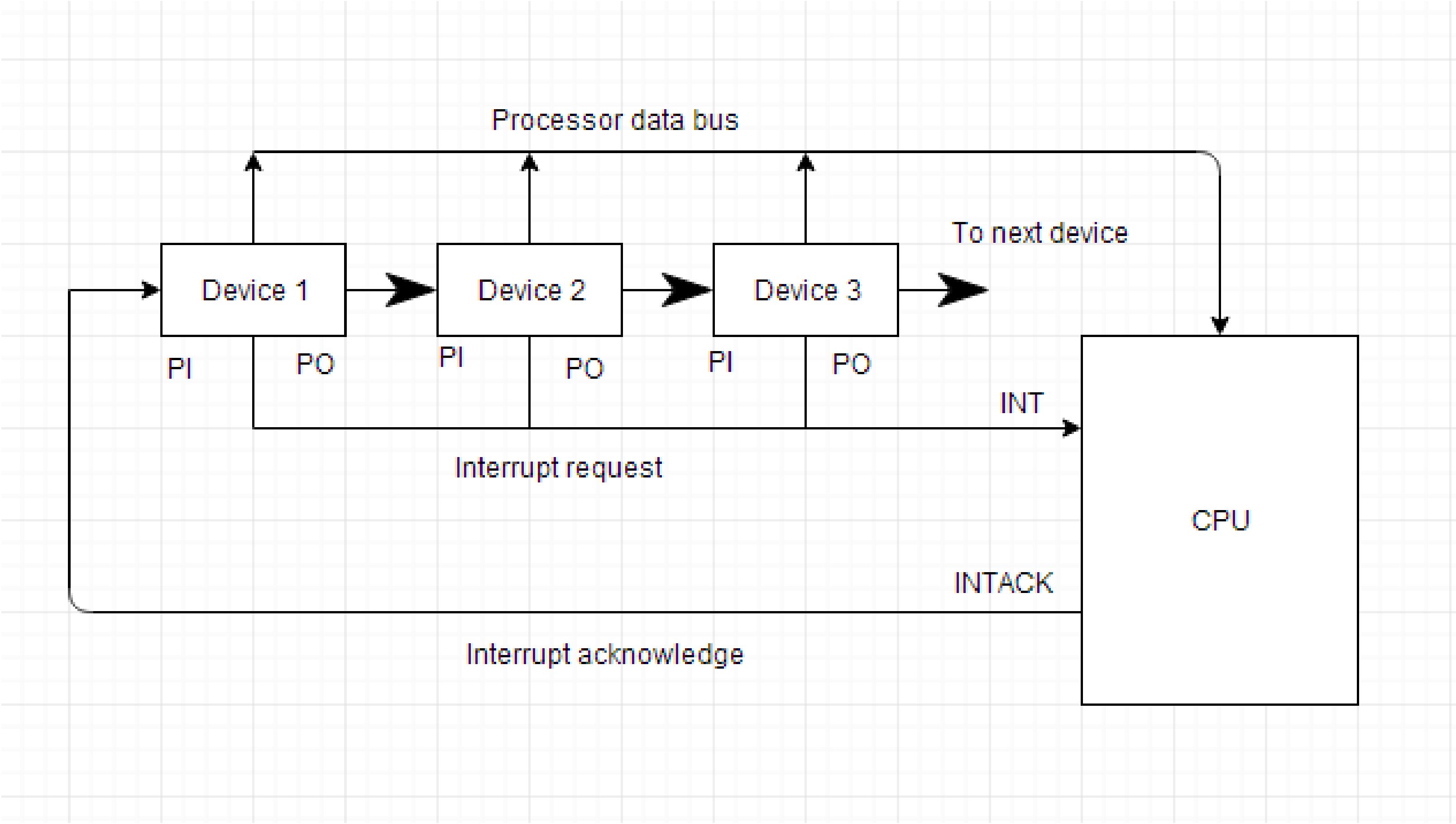
Device Identification-2

- Hardware Poll
    - All I/O modules share a common interrupt request line
    - Interrupt ACK line is Daisy changed through the modules
    - Requesting module places address of the I/O module on data bus which is called as a vector
    - Technique is called as vectored Interrupt

Daisy Chaining

- This way of deciding the interrupt priority consists of serial connection of all the devices which generates an interrupt signal.
- The device with the highest priority is placed at the first position followed by lower priority devices and the device which has lowest priority among all is placed at the last in the chain.

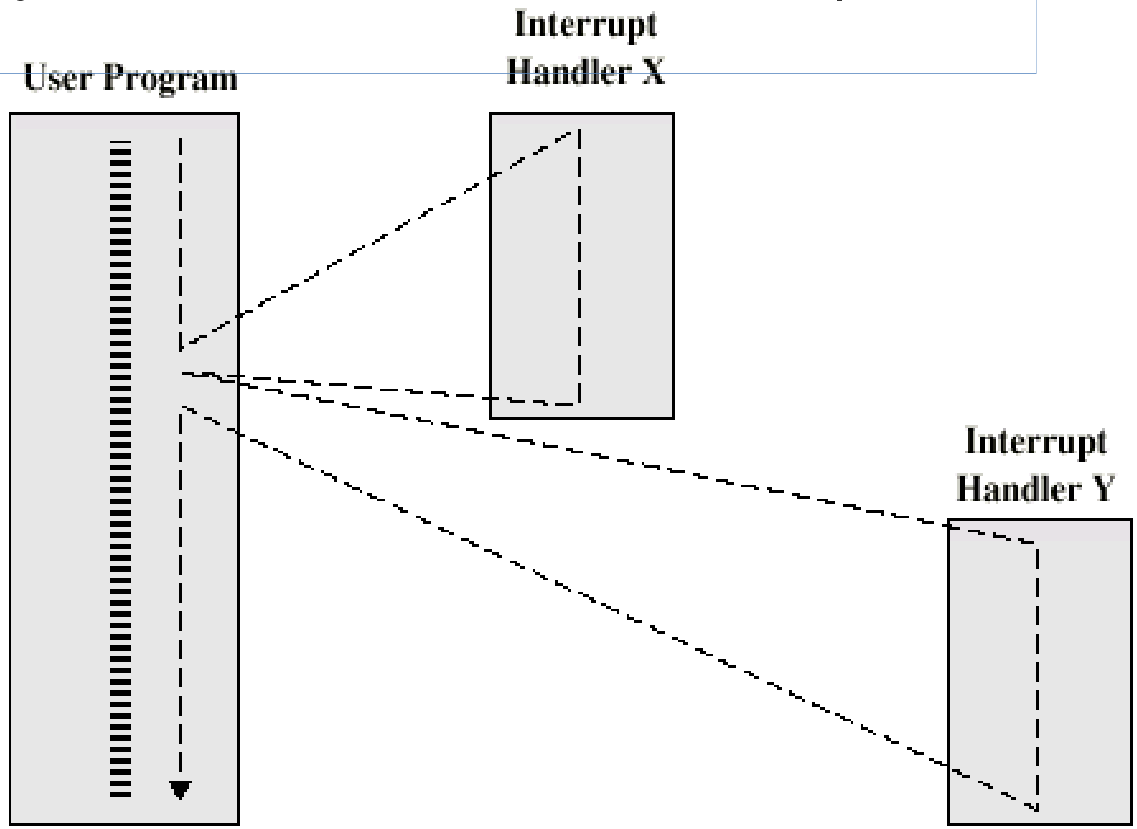
Daisy Chaining


Processor data bus
To next device
Device 1
Device 2
Device 3
PI
PO
PI
PO
PI
PO
INT
Interrupt request
CPU
INTACK
Interrupt acknowledge

Multiple Interrupts - Sequential

Processor will ignore further interrupts whilst processing one interrupt

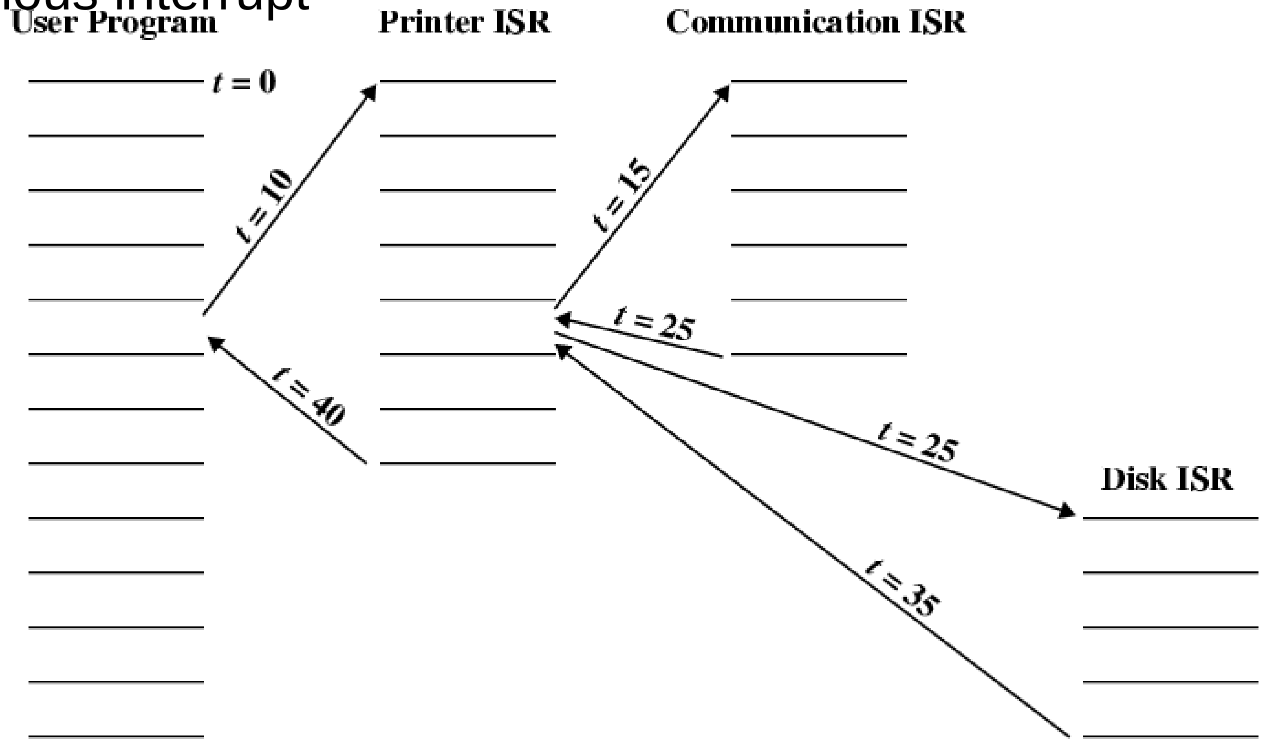
Interrupts remain pending and are checked after first interrupt has been processed


User Program
Interrupt Handler X
Interrupt Handler Y

Multiple Interrupts: Priority

Low priority interrupts can be interrupted by higher priority interrupts

When higher priority interrupt has been processed, processor returns to previous interrupt


User Program
Printer ISR
Communication ISR
Disk ISR
t = 0
t = 10
t = 15
t = 25
t = 25
t = 35
t = 40

Review Questions

- When a device interrupt occurs, how does the processor determine which device issued the interrupt?
- What is context switching?
- How programmed I/O is different from Interrupt Driven I/O?
- How software polling is different from Daisy chain method to handle multiple interrupts?

PROGRAMMED I/O

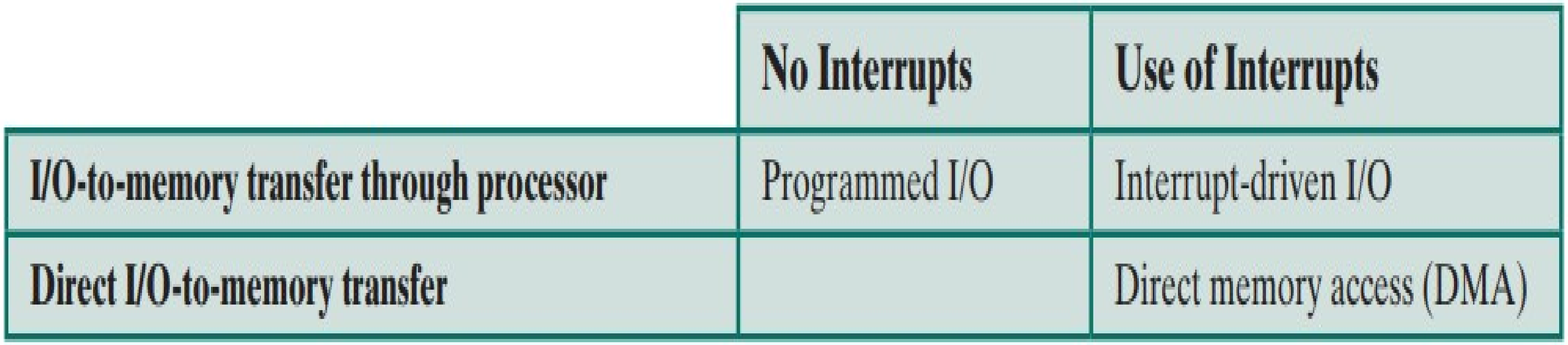
- Data are exchanged between the processor and the I/O module.

- Direct control of the I/O operation. Sensing device status, sending a read or write command, and transferring the data.

- With interrupt-driven I/O, the processor issues an I/O command, executes other instructions, and is interrupted by the I/O module when the latter has completed its work.

- **DMA** - With both programmed and interrupt I/O, the processor extracts data from main memory for output and stores data in main memory for input.

I/O Techniques

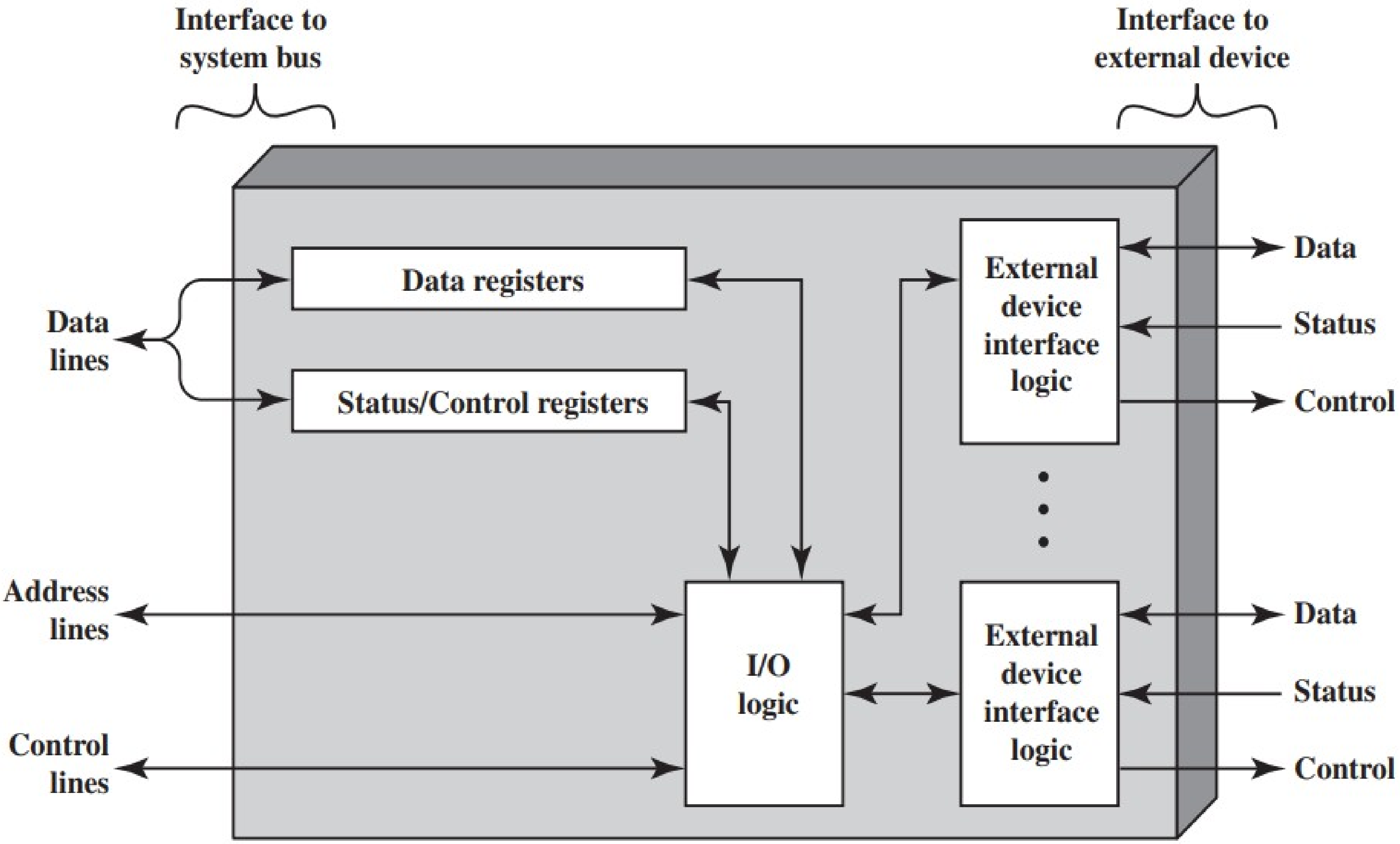


	No Interrupts	Use of Interrupts
I/O-to-memory transfer through processor	Programmed I/O	Interrupt-driven I/O
Direct I/O-to-memory transfer		Direct memory access (DMA)

Overview of Programmed I/O

- The I/O module will perform the requested action and then set the appropriate bits in the I/O status register.
- I/O module does not interrupt the processor.
- It periodically checks the status of the I/O module until it finds that the operation is complete.

Block Diagram of an I/O Module


Interface to system bus
Interface to external device
Data lines
Address lines
Control lines
Data registers
Status/Control registers
I/O logic
External device interface logic
External device interface logic
Data
Status
Control
Data
Status
Control

I/O Commands

Four types of I/O commands:

- **Control**
- **Test**
- **Read**
- **Write**

I/O Instructions

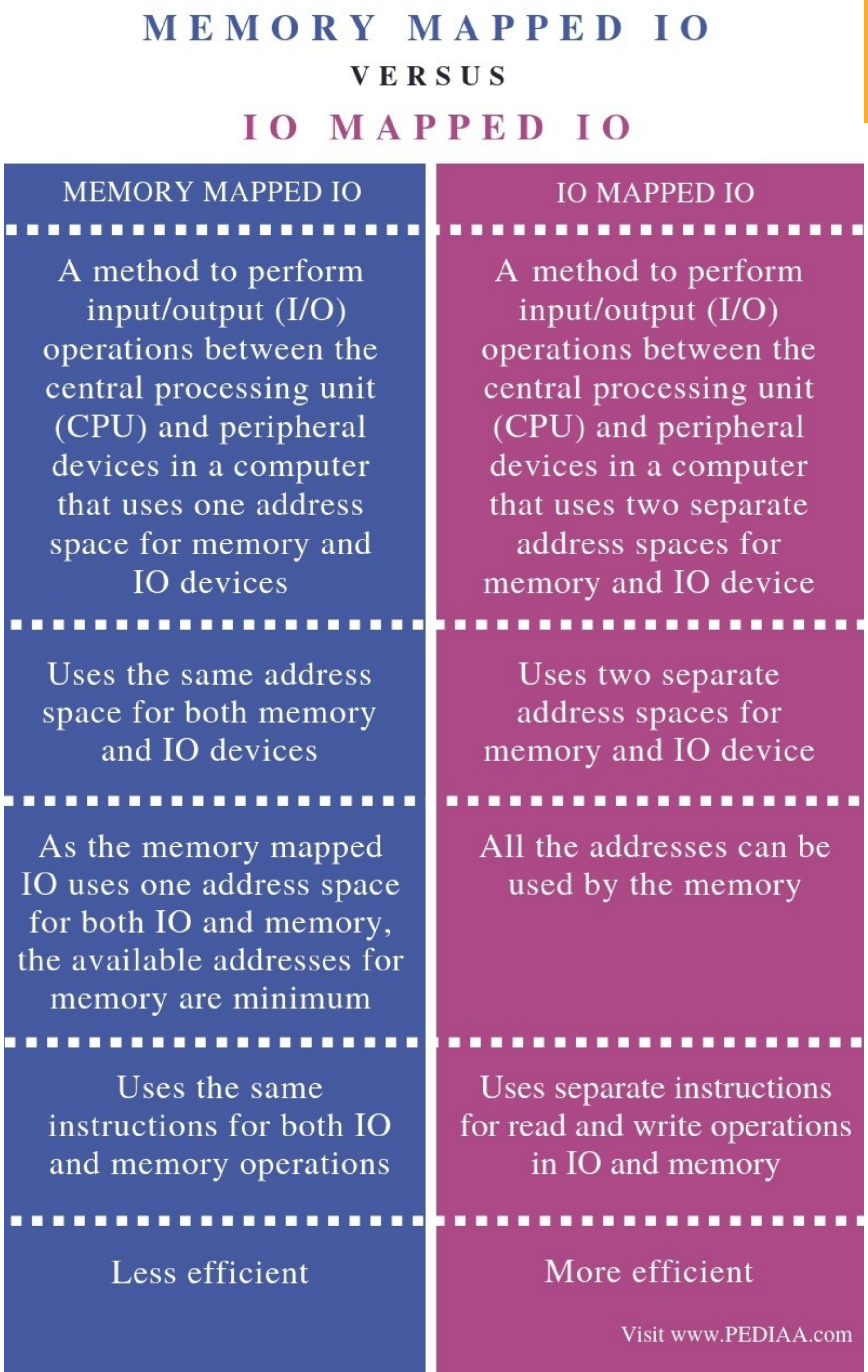
- The instructions are easily mapped into I/O commands

- A simple one-to-one relationship

- The form of the instruction depends on the way in which external devices are addressed

- **Memory- mapped I/O:** single address space for memory locations and I/O devices, same machine instructions to access both memory and I/O devices, a single read line and a single write line are needed on the bus.

- **Isolated I/O:** full range of addresses may be available for both, , the bus may be equipped with memory read and write plus input and output command lines, the bus may be equipped with memory read and write plus input and output command lines.



MEMORY MAPPED IO
VERSUS
IO MAPPED IO

MEMORY MAPPED IO	IO MAPPED IO
A method to perform input/output (I/O) operations between the central processing unit (CPU) and peripheral devices in a computer that uses one address space for memory and IO devices	A method to perform input/output (I/O) operations between the central processing unit (CPU) and peripheral devices in a computer that uses two separate address spaces for memory and IO device
Uses the same address space for both memory and IO devices	Uses two separate address spaces for memory and IO device
As the memory mapped IO uses one address space for both IO and memory, the available addresses for memory are minimum	All the addresses can be used by the memory
Uses the same instructions for both IO and memory operations	Uses separate instructions for read and write operations in IO and memory
Less efficient	More efficient

Visit www.PEDIAA.com

INTERRUPT-DRIVEN I/O

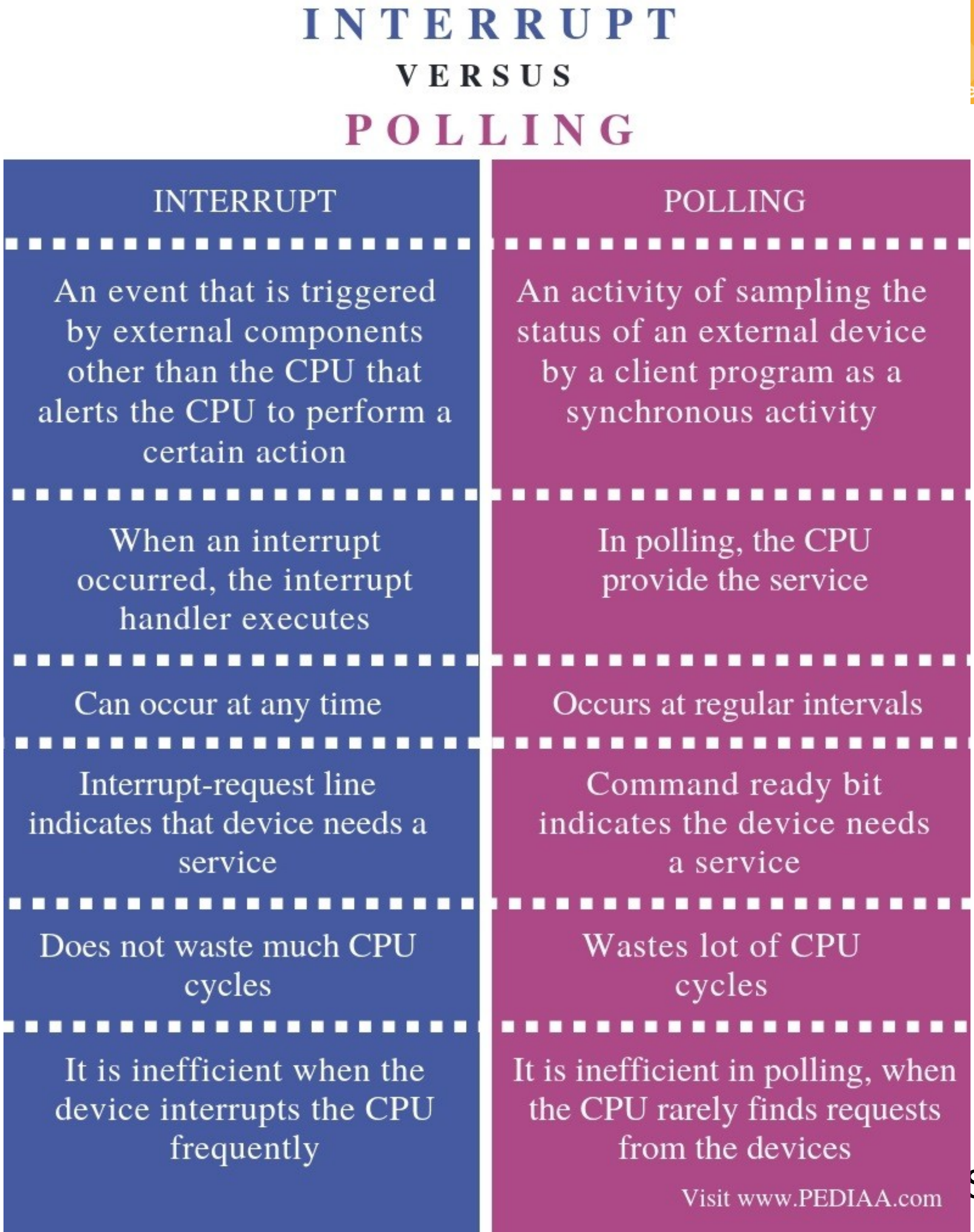
- The processor issues a READ command and then it goes off. At the end of each instruction cycle, the processor checks for interrupts. When interrupts occurs, the processor saves the context of the current program and processes the interrupt. The processor reads the word of data from the I/O module and stores it in memory. It then restores the context of the program it was working on and resumes execution.

- Interrupt I/O is **more efficient** because it eliminates needless waiting.

- **Disadvantage of Interrupt I/O**: consumes a lot of processor time (every word of data that goes from memory to I/O module or from I/O module to memory must pass through the processor)

Design Issues

- Two design issues arise in implementing interrupt I/O.
- **How does the processor determine which device issued the interrupt?**
- **If multiple interrupts have occurred, how does the processor decide which one to process?**
- Four general categories of techniques are in common use:
  - Multiple interrupt lines
  - Software poll
  - Daisy chain (hardware poll, vectored)
  - Bus arbitration (vectored)



INTERRUPT
VERSUS
POLLING

INTERRUPT	POLLING
An event that is triggered by external components other than the CPU that alerts the CPU to perform a certain action	An activity of sampling the status of an external device by a client program as a synchronous activity
When an interrupt occurred, the interrupt handler executes	In polling, the CPU provide the service
Can occur at any time	Occurs at regular intervals
Interrupt-request line indicates that device needs a service	Command ready bit indicates the device needs a service
Does not waste much CPU cycles	Wastes lot of CPU cycles
It is inefficient when the device interrupts the CPU frequently	It is inefficient in polling, when the CPU rarely finds requests from the devices

Visit www.PEDIAA.com

Drawbacks of Interrupt Driven and Programmed I/O

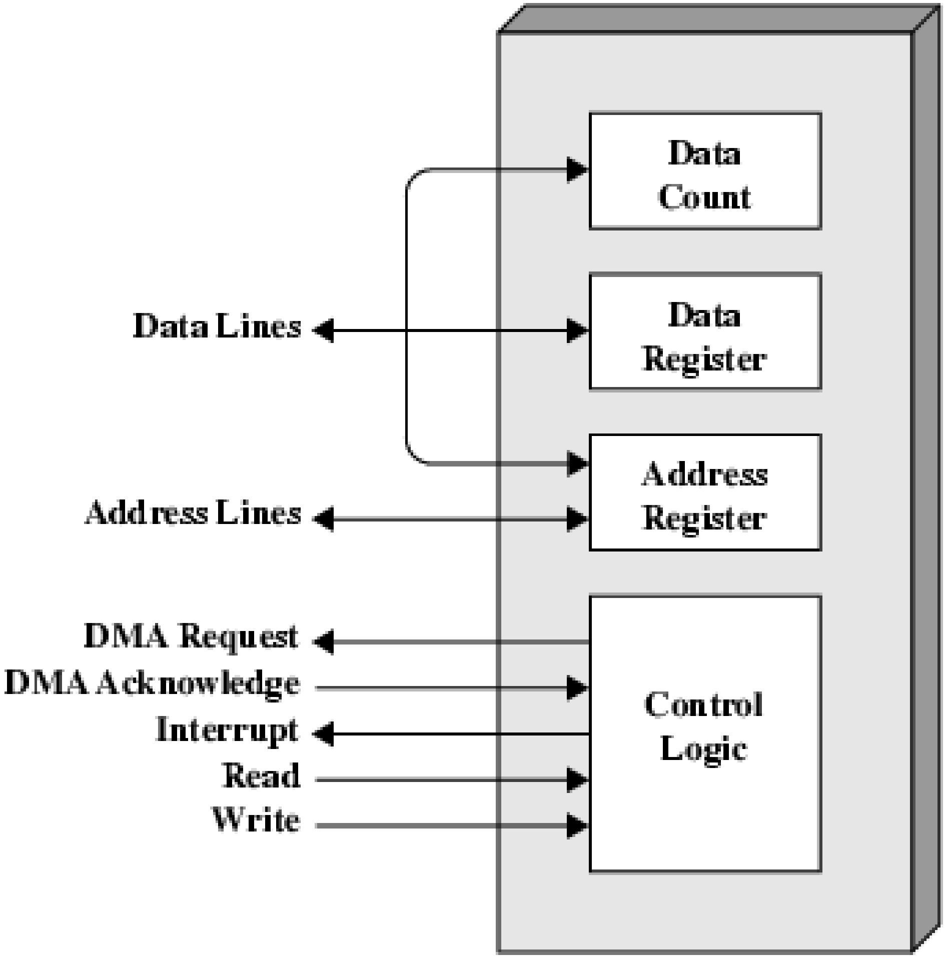
- Requires active CPU Intervention
  - Transfer rate is limited by the speed with which the processor can test and service the device
  - CPU is tied up in managing an I/O transfer
- Using programmed I/O processor can move data at higher rate at the cost of doing nothing else
- Interrupt I/O frees up the processor at the expense of the I/O transfer rate

Direct Memory Access (DMA)

- A special control unit can be provided to allow transfer of a block of data directly between an external device and the main memory.
- No continuous intervention by the processor is required
- This control circuit can be part of I/O device interface and called as DMA controller
- Good for moving large volume of data

DMA Module and Operation

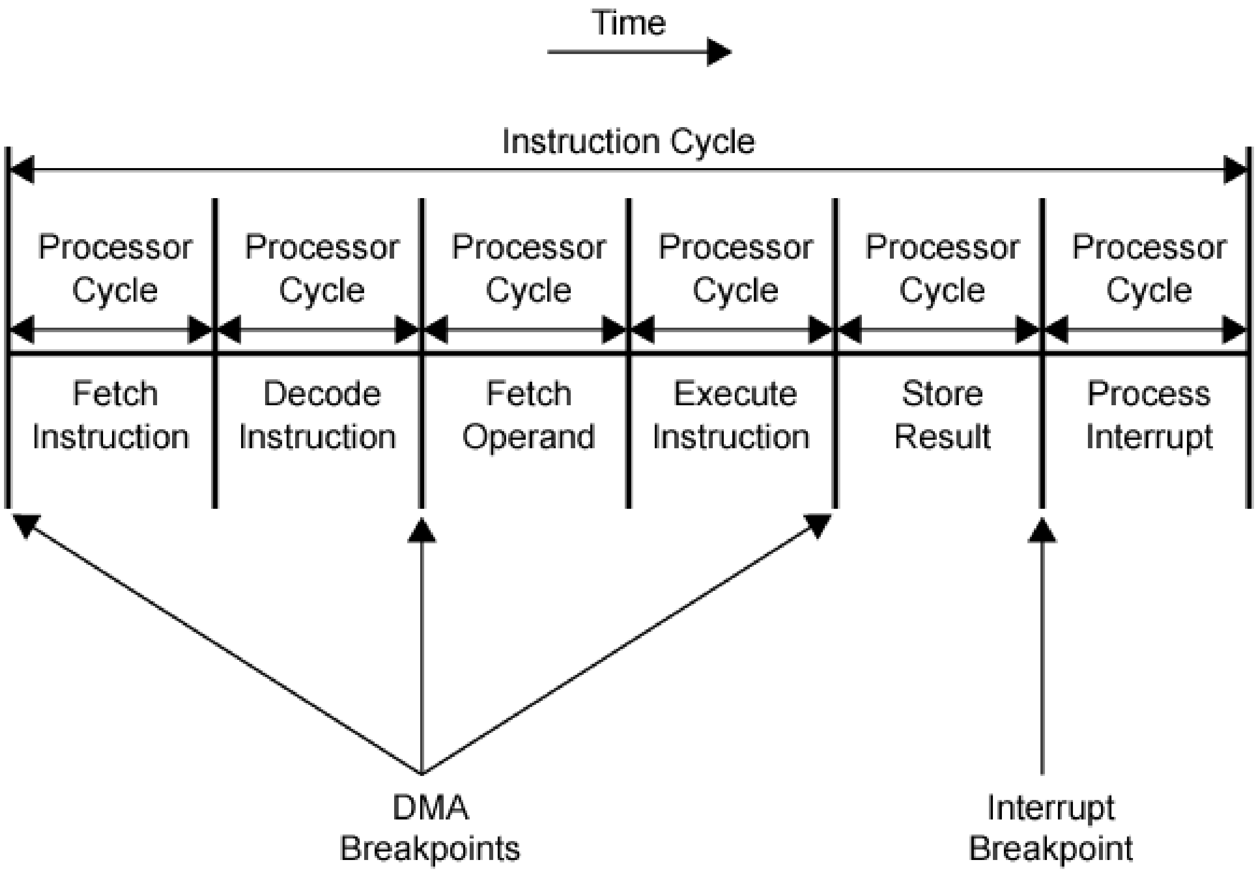
- CPU tells DMA controller:-
  - Read/Write
  - Device address
  - Starting address of memory block for data
  - Amount of data to be transferred
- CPU carries on with other work
- DMA controller deals with transfer
- DMA controller sends interrupt when finished


Data Count
Data Register
Address Register
Control Logic
Data Lines
Address Lines
DMA Request
DMA Acknowledge
Interrupt
Read
Write

DMA Transfer Method: Cycle Stealing

- Memory accesses by the processor and by the DMA controllers are interwoven
- DMA controller takes over bus for a cycle
- Transfer of one word of data
- Not an interrupt
  - CPU does not switch context
- CPU suspended just before it accesses bus
  - i.e. before an operand or data fetch or a data write
- Slows down CPU but not as much as CPU itself doing transfer

DMA and Interrupt Breakpoints During an Instruction Cycle



Time

Instruction Cycle

Processor Cycle	Processor Cycle	Processor Cycle	Processor Cycle	Processor Cycle	Processor Cycle
Fetch Instruction	Decode Instruction	Fetch Operand	Execute Instruction	Store Result	Process Interrupt

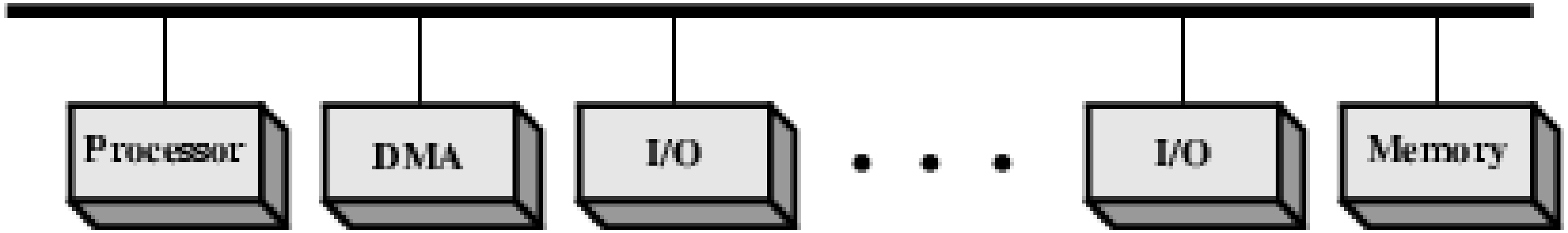
DMA Breakpoints

Interrupt Breakpoint

DMA Transfer Method: Burst Mode

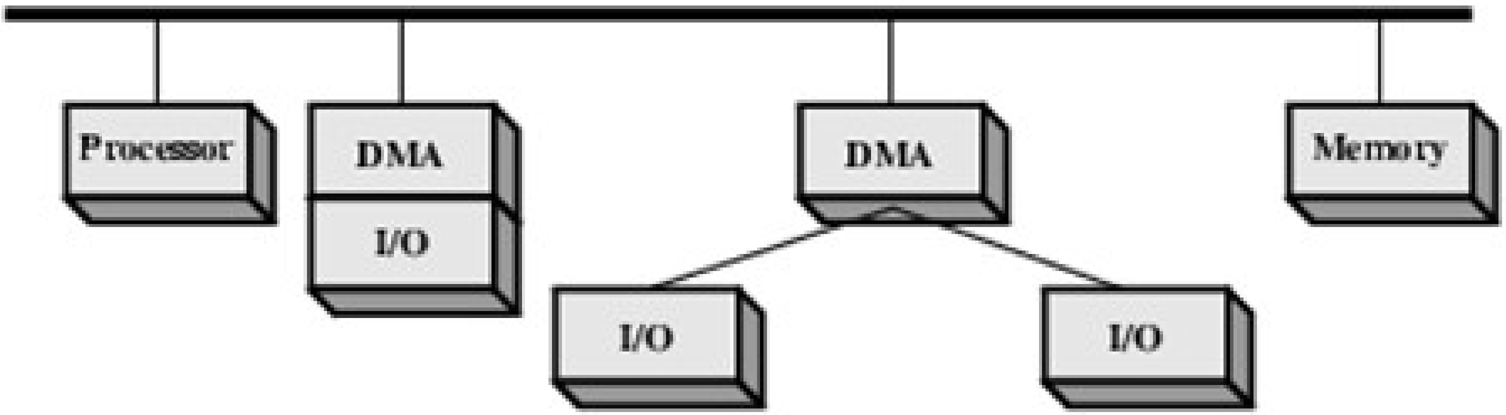
- DMA controller is given exclusive access to the main memory to transfer a block of data without interruption
- Most DMA controllers incorporate a data storage buffer
- It reads a block of data using burst mode from the main memory and stores it into its input buffer
- Then it is transferred to the desired I/O device

DMA Configurations (1)


Processor
DMA
I/O
• • •
I/O
Memory

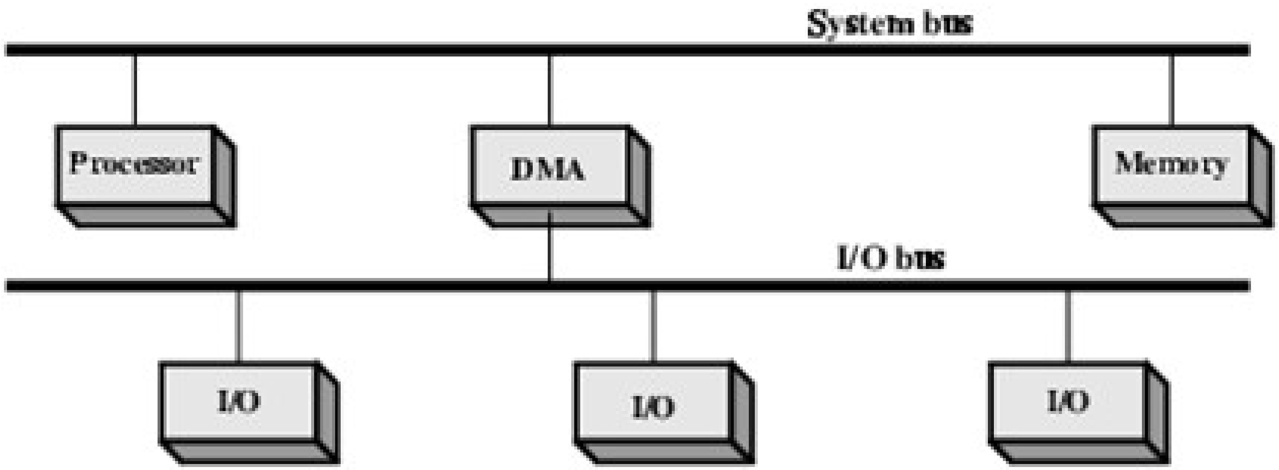
- Single Bus, Detached DMA controller
- Each transfer uses bus twice
  - I/O to DMA then DMA to memory
- CPU is suspended twice!

DMA Configurations (2)


Processor
DMA
I/O
DMA
Memory
I/O
I/O

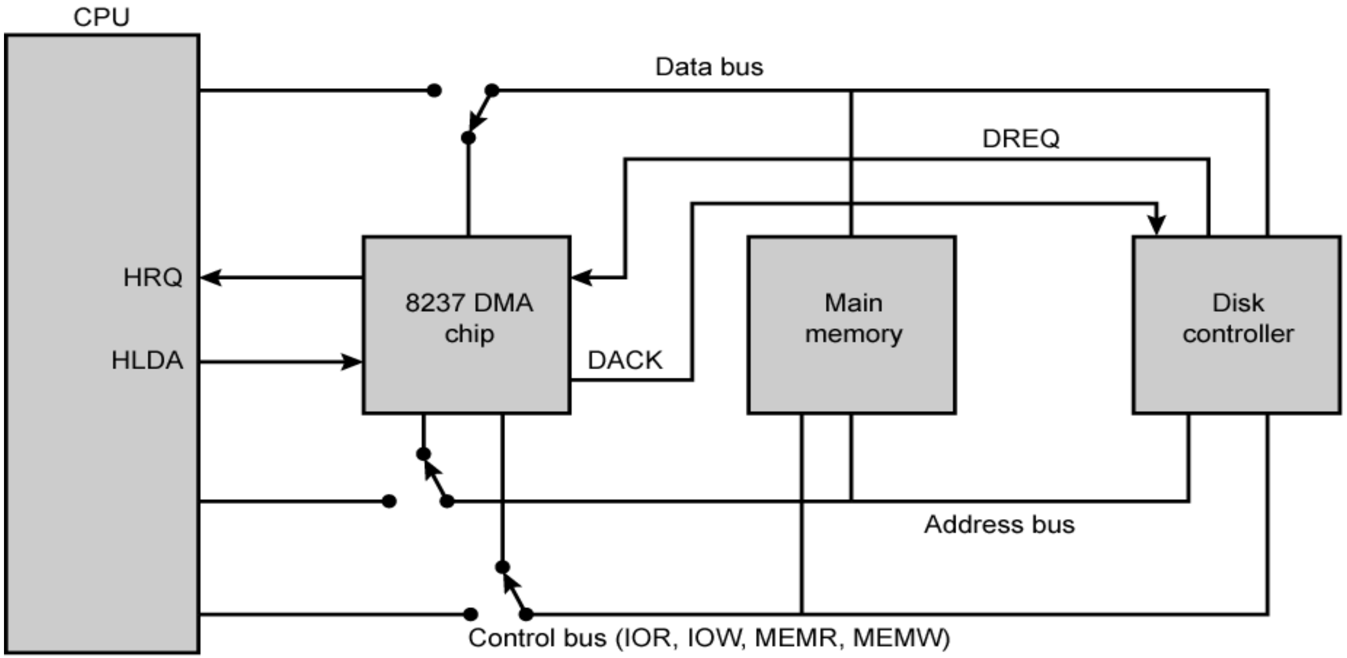
- Single Bus, Integrated DMA controller
- Controller may support >1 device
- Each transfer uses bus once
- CPU is suspended once!

DMA Configurations (3)


System bus
Processor
DMA
Memory
I/O bus
I/O
I/O
I/O

- Separate I/O Bus
- Bus supports all DMA enabled devices
- Each transfer uses system bus once
- CPU is suspended once

Intel 8237A DMA Controller

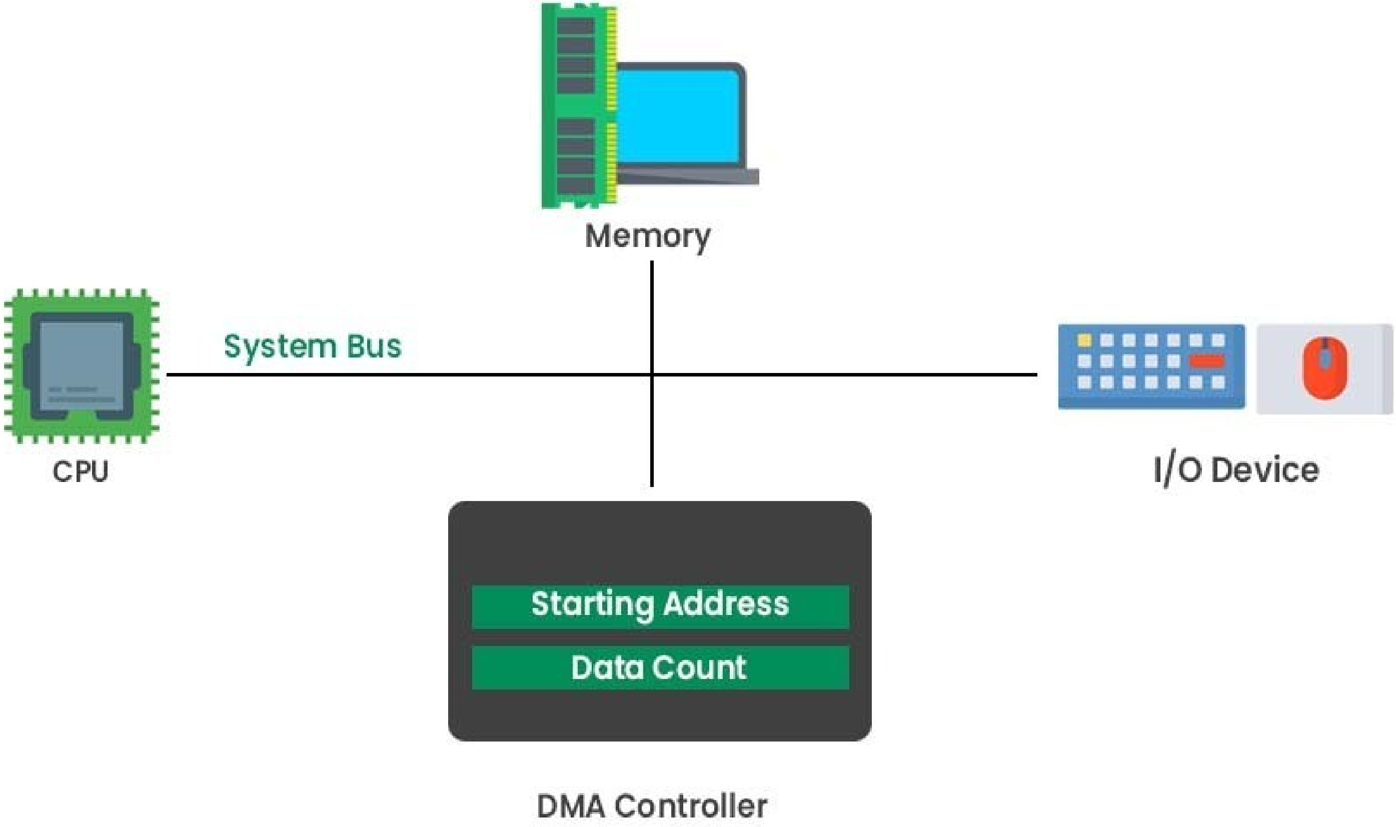

CPU
Data bus
DREQ
HRQ
HLDA
8237 DMA chip
DACK
Main memory
Disk controller
Address bus
Control bus (IOR, IOW, MEMR, MEMW)

DACK = DMA acknowledge
DREQ = DMA request
HLDA = HOLD acknowledge
HRQ = HOLD request

- DMA basically stands for Direct Memory Access.
- It is a process which enables data transfer between the Memory and the IO (Input/ Output) device without the need of or you can say without the involvement of CPU during data transfer.

Working of DMA

- For DMA, you basically need a hardware called DMAC (Direct Memory Access Controller) which will help in the throughout process of data transfer between the Memory and IO device directly.
- First what happens is IO device sends the DMA request to DMA Controller, then further DMAC device sends HOLD signal to CPU by which it asks CPU for several information which are needed while transferring data.
- CPU then shares two basic information with DMAC before the Data transfer which are: Starting address (memory address starting from where data transfer should be performed) and Data Count (no of bytes or words to be transferred).
- CPU then sends HLDACK (Hold Acknowledgement) back to DMAC illustrating that now DMAC can successfully pass on the information.
- Then further DMAC shares the DMA ACK (DMA Acknowledgement) to the IO device which would eventually let IO device to access or transfer the data from memory in a direct and efficient manner.


Memory
System Bus
CPU
I/O Device
Starting Address
Data Count
DMA Controller

Modes of DMA Transfer :

- Now after getting some brief idea about DMA and its working it's the time to analyze Modes of DMA Transfer.
- During the DMA Transfer CPU can perform only those operation in which it doesn't require the access of System Bus which means mostly CPU will be in blocked state.
- For how much time CPU remains in the blocked state or we can say for how much time CPU will give the control of DMAC of system buses will actually depend upon the following modes of DMA Transfer and after that CPU will take back control of system buses from DMAC.
- Burst mode, cycle stealing mode and interleaving mode

Burst Mode

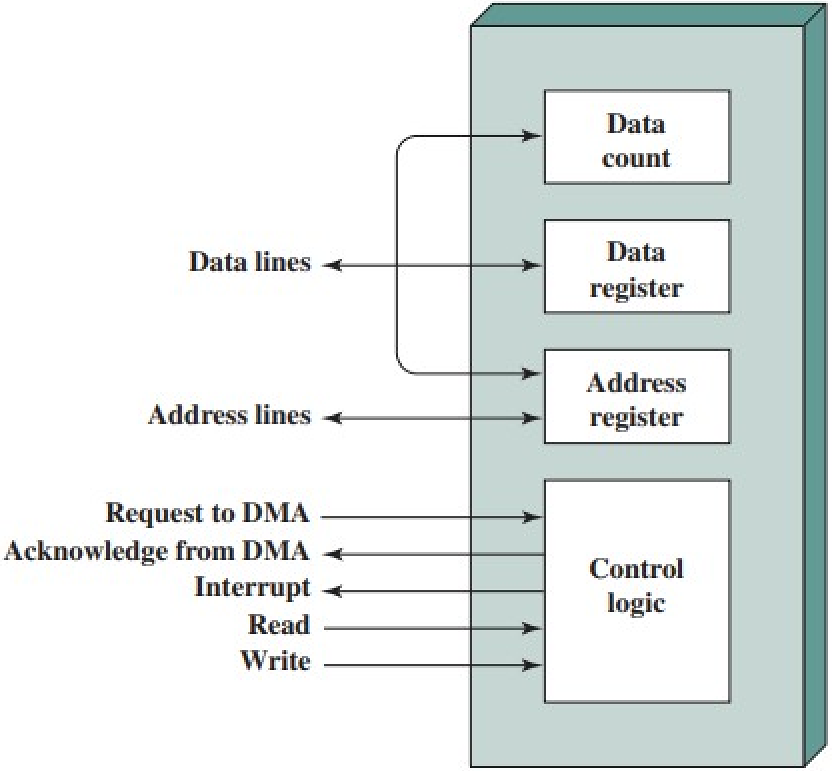
- In this mode Burst of data (entire data or burst of block containing data) is transferred before CPU takes control of the buses back from DMAC.
- This is the quickest mode of DMA Transfer since at once a huge amount of data is being transferred.
- Since at once only the huge amount of data is being transferred so time will be saved in huge amount.

DIRECT MEMORY ACCESS (DMA)

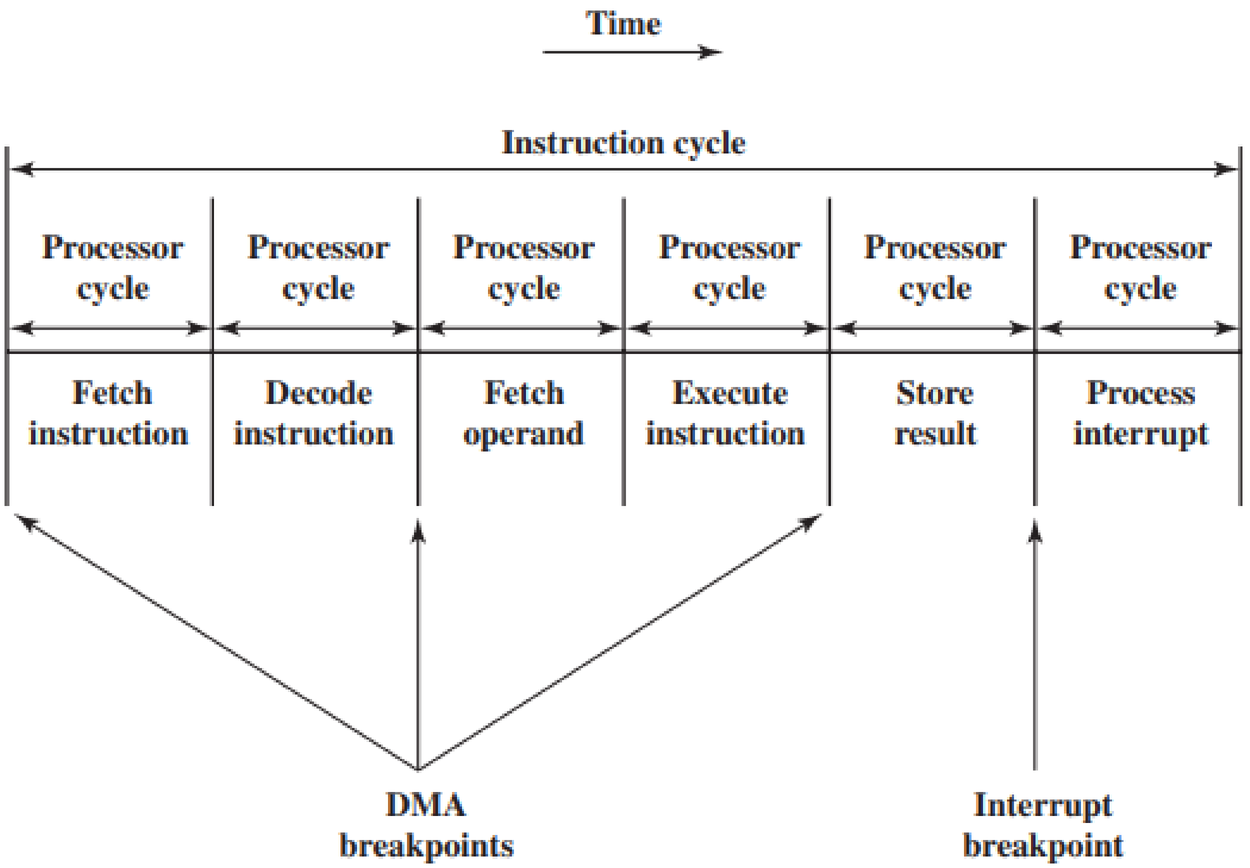
- Drawbacks of Programmed and Interrupt-Driven I/O:
  - The I/O transfer rate is limited by the speed with which the processor can test and service a device.
  - The processor is tied up in managing an I/O transfer; a number of instructions must be executed for each I/O transfer.
- When large volumes of data are to be moved, a more efficient technique is required: direct memory access (DMA).

DMA Function

- An additional module on the system bus
- Capable of mimicking the processor
- **Cycle stealing** - the DMA module in effect steals a bus cycle


Data count
Data lines
Data register
Address lines
Address register
Request to DMA
Acknowledge from DMA
Interrupt
Read
Write
Control logic

DMA and Interrupt Breakpoints during an Instruction Cycle


Time
Instruction cycle
Processor cycle
Processor cycle
Processor cycle
Processor cycle
Processor cycle
Processor cycle
Fetch instruction
Decode instruction
Fetch operand
Execute instruction
Store result
Process interrupt
DMA breakpoints
Interrupt breakpoint

- The processor is suspended just before it needs to use the bus.
- The DMA module transfers one word and returns control to the processor.
- The processor pauses for one bus cycle. The overall effect is to cause the processor to execute more slowly.
- For a multiple-word I/O transfer, DMA is more efficient than interrupt-driven or programmed I/O.

THANK YOU

BITS Pilani
Pilani Campus

BITS Pilani
Pilani Campus

COMPUTING AND DESIGN

SESSION 10

Course design by - Dr. Lucy J. Gudino &
Dr Chandrasekhar
Faculty – Dr Thangakumar J
WILP & Department of CS & IS

Operating systems – I/O Abstraction (cont..)

BITS Pilani
Pilani Campus

Last Session

List of Topic Title	Text/Ref Book/external resource
• I/O System Hardware o Device Controllers o Types of devices : Terminal Devices, Communication Devices and Disk Devices • Data transfer schemes • Programmed I/O, Interrupt Driven I/O, DMA	T1 (14.1 – 14.4) R6 (7.3-7.5)

INPUT OUTPUT – STORY LINE

1

POLLING

- CHECK
- INITIATE
- COMPLETE

2

- INTERRUPTIONS
  - SAVE CONTEXT IN MEMORY
  - INTERRUPT SERVICE ROUTINE
  - LOOKUP TO EXECUTE INTERRUPT SERVICE ROUTINE
  - RESTORE CONTEXT AND RESUME OPERTIONS

3

DIRECT MEMORY ACCESS

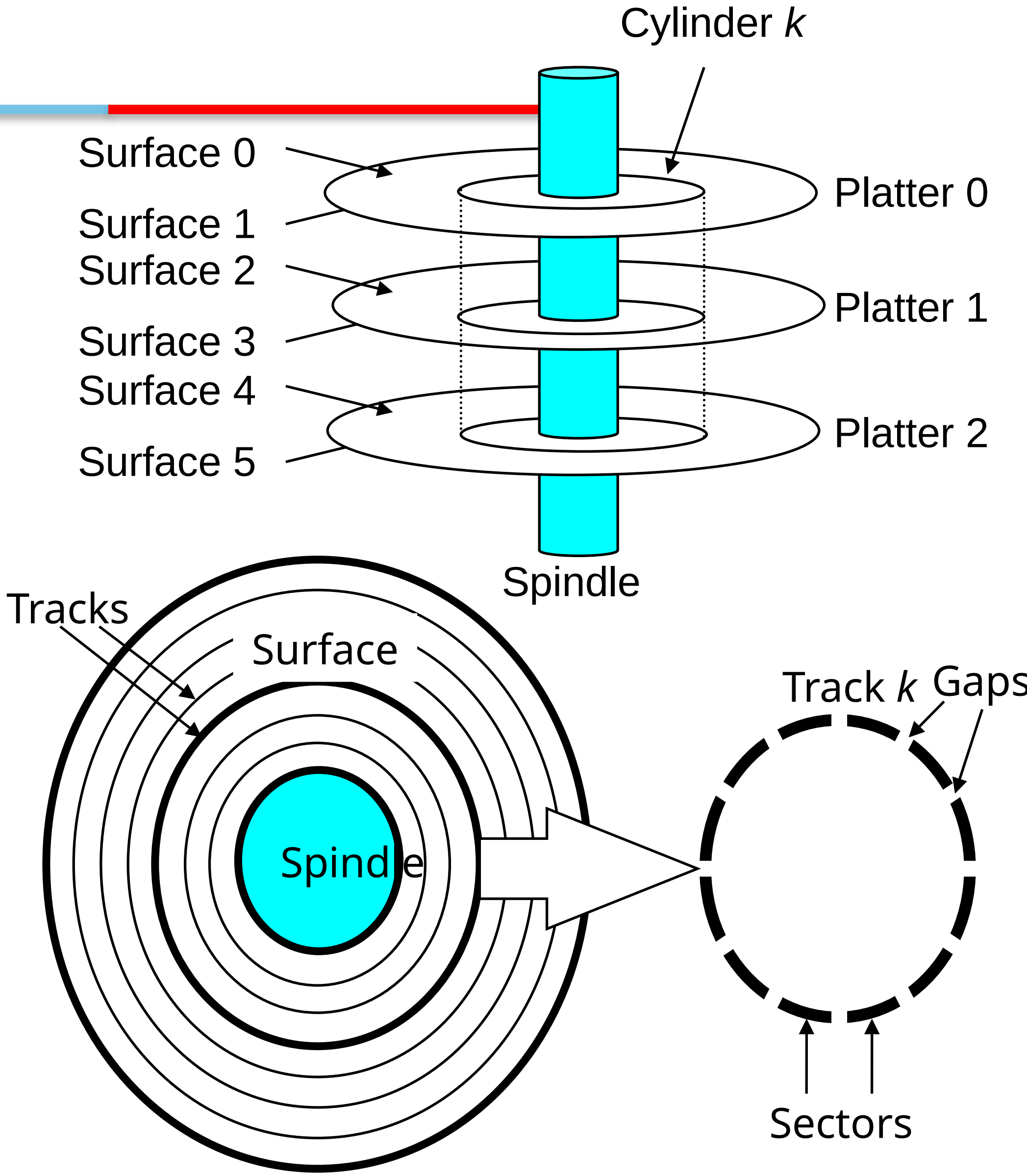
- INPUT OUTPUT IN BLOCKS
- WHICH DEVICE, HOW MANY BLOCKS, MEMORY LOCATION.
- CPU “interrupted” to check for errors; assigns next program

4

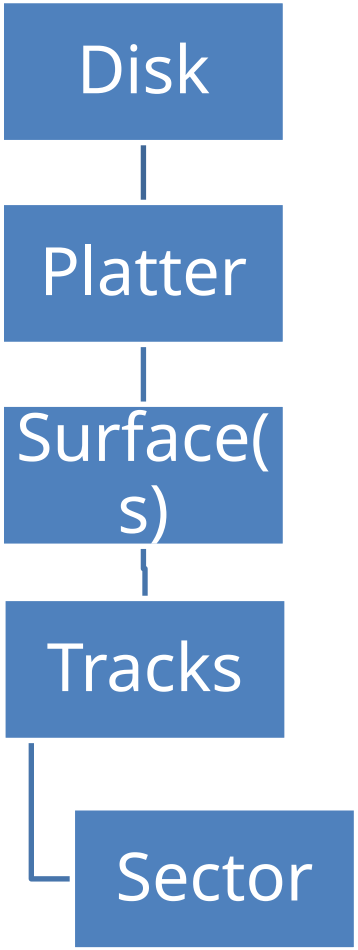
- INPUT OUTPUT PROCESSORS
  - IOP CAN MANAGE MANY DEVICES
  - IOP CAN MANAGE HIGH LEVEL IOP
  - MORE SOPHISTICATED THAN DMA

Disk Geometry

- Disks consist of platters, each with two surfaces.
- Each surface consists of concentric rings called tracks
- Aligned tracks form a cylinder
- Each track consists of sectors separated by gaps.


Cylinder *k*
Surface 0
Surface 1
Surface 2
Surface 3
Surface 4
Surface 5
Platter 0
Platter 1
Platter 2
Spindle
Tracks
Surface
Spindle
Track *k*
Gaps
Sectors

Disk Geometry (contd)



Disk

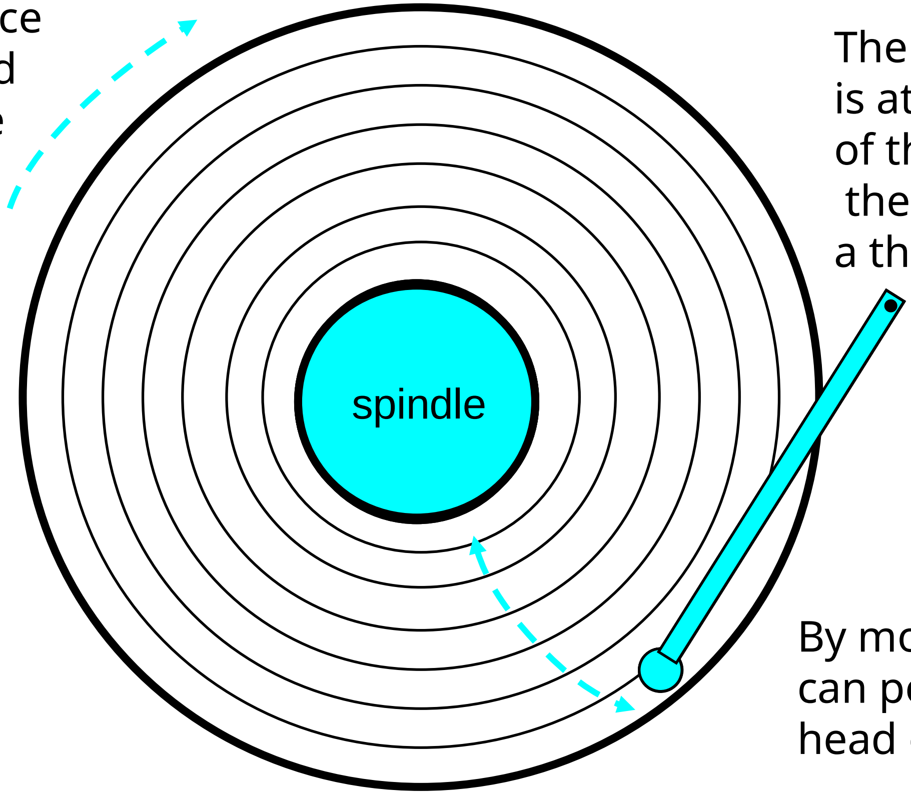
Platter

Surface(s)

Tracks

Sector

Disk Operation (Single-Platter View)

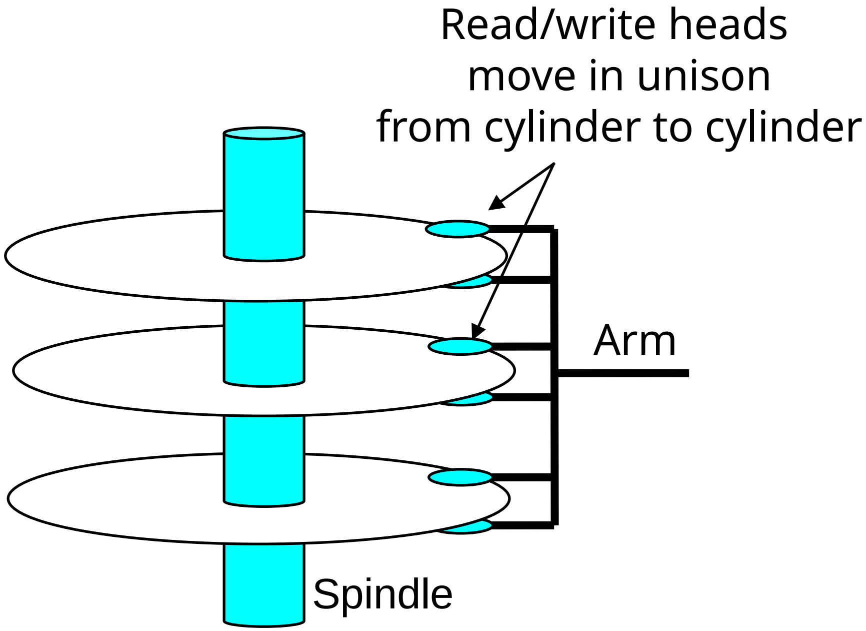


The disk surface
spins at a fixed
rotational rate

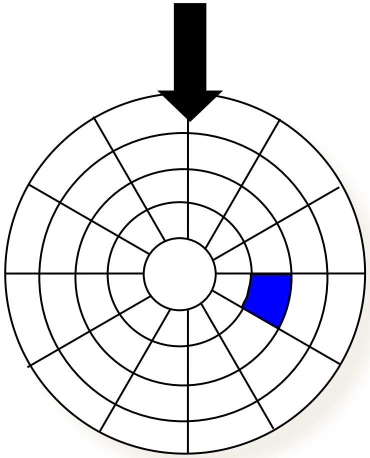
The read/write *head*
is attached to the end
of the *arm* and flies over
the disk surface on
a thin cushion of air.

By moving radially, the arm
can position the read/write
head over any track.

Disk Operation (Multi-Platter View)

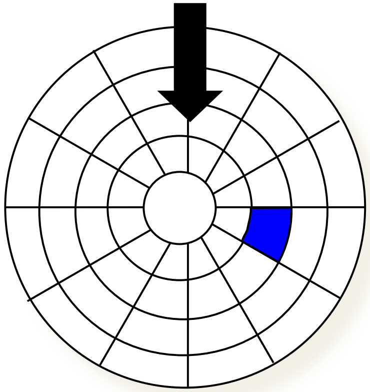

Read/write heads
move in unison
from cylinder to cylinder
Arm
Spindle

Disk Access



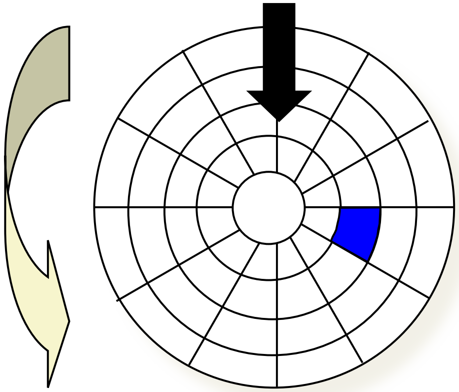
Need to access a sector colored in blue

Disk Access



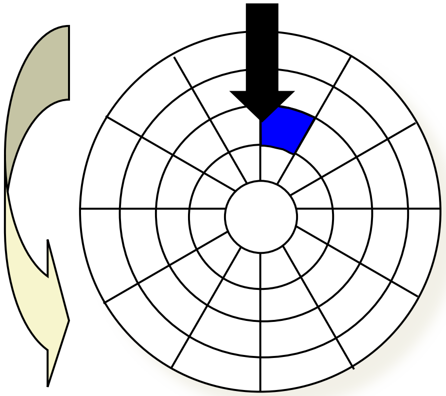
Head in position above a track

Disk Access



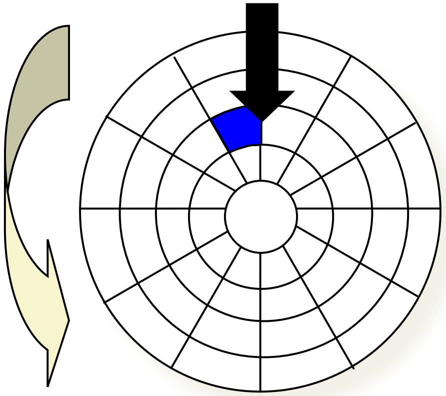
Rotate the platter in counter-clockwise direction

Disk Access – Read



About to read blue sector

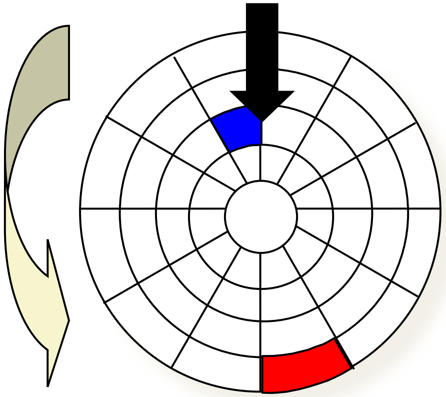
Disk Access – Read



After BLUE read

After reading blue sector

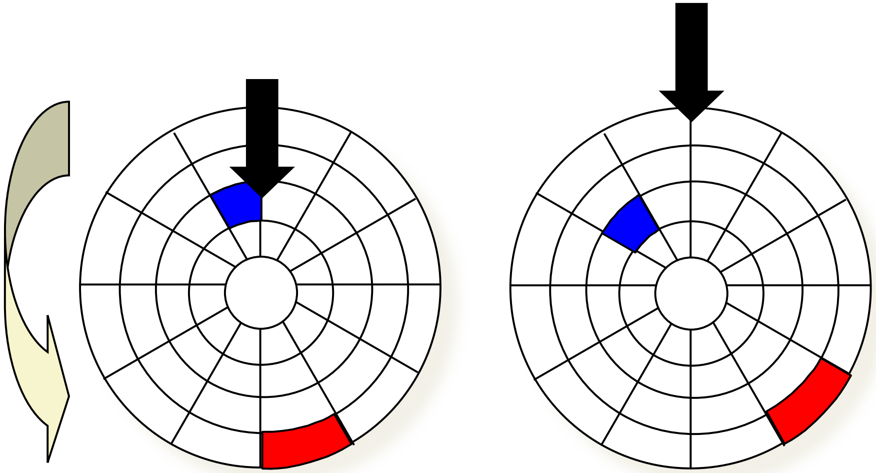
Disk Access – Read



After BLUE read

Red request scheduled next

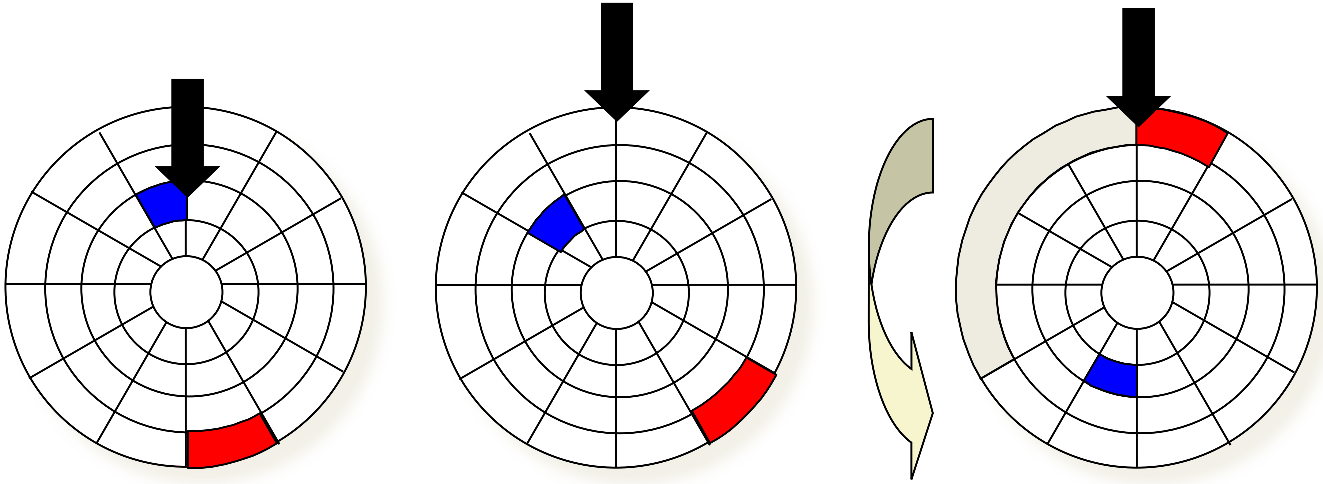
Disk Access – Seek



After BLUE read   Seek for RED

Seek to red’s track

Disk Access – Rotational Latency



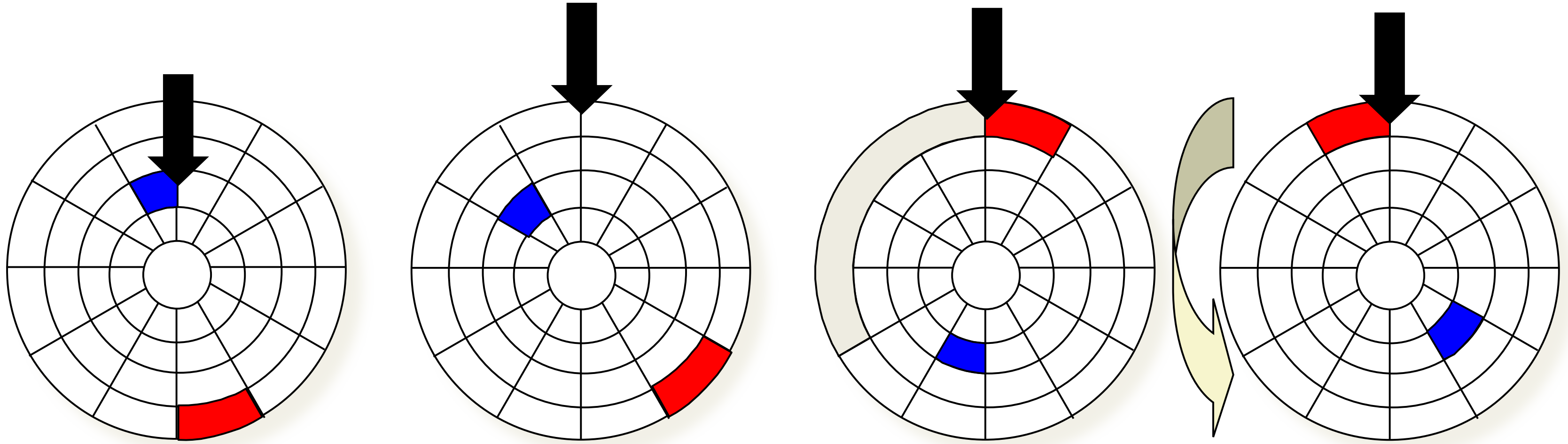
After BLUE read

Seek for RED

Rotational latency

Wait for red sector to rotate around

Disk Access – Read



After BLUE read

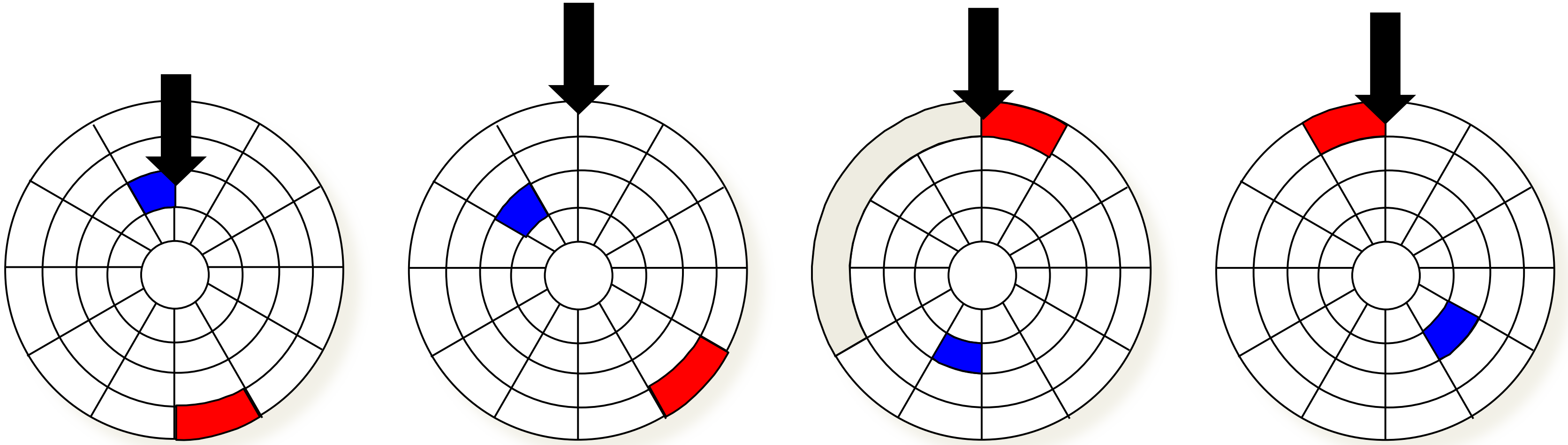
Seek for RED

Rotational latency

After RED read

Complete read of red

Disk Access – Access Time Components



After BLUE read

Seek for RED

Rotational latency

After RED read

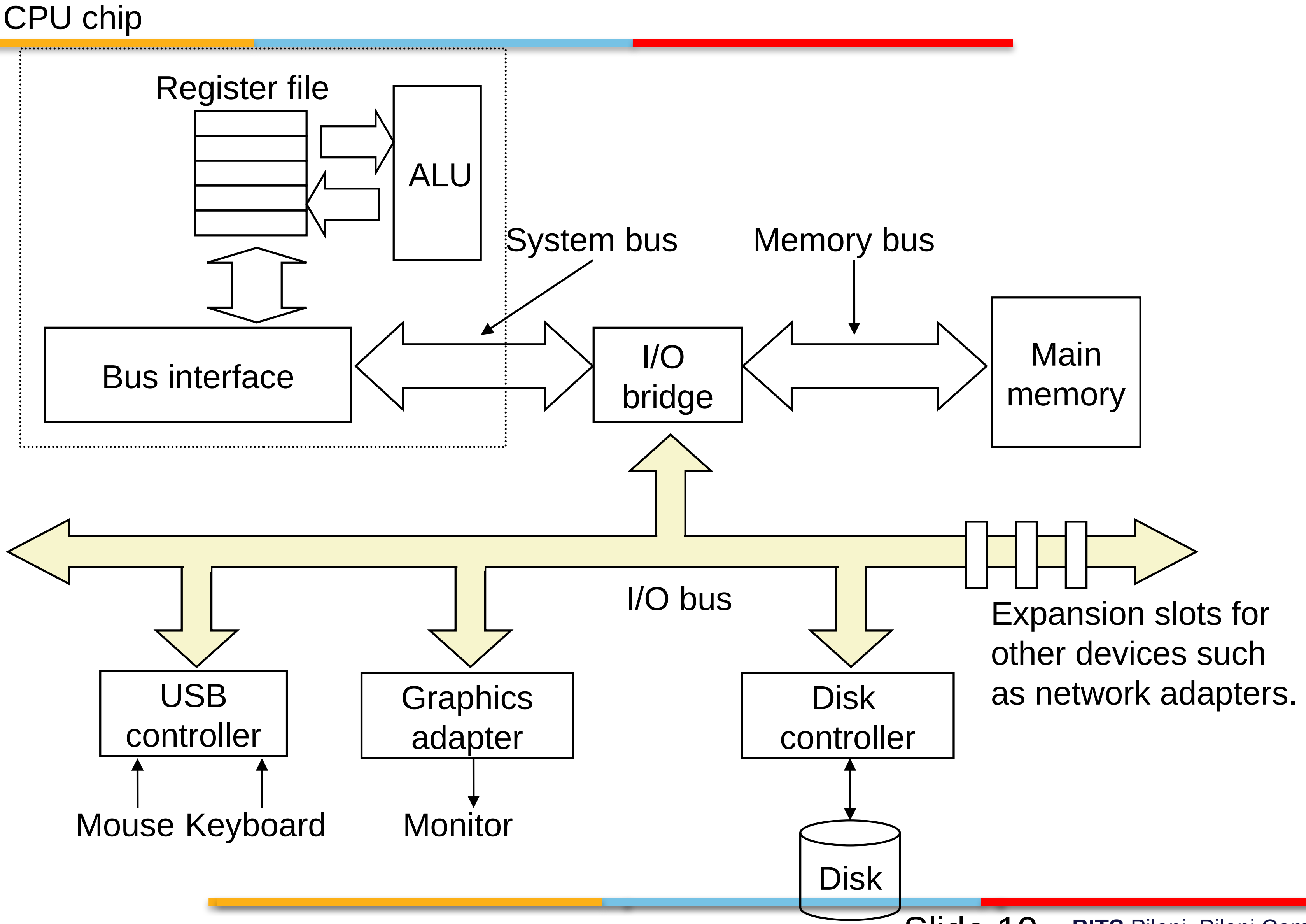
↑ Data transfer

↑ Seek

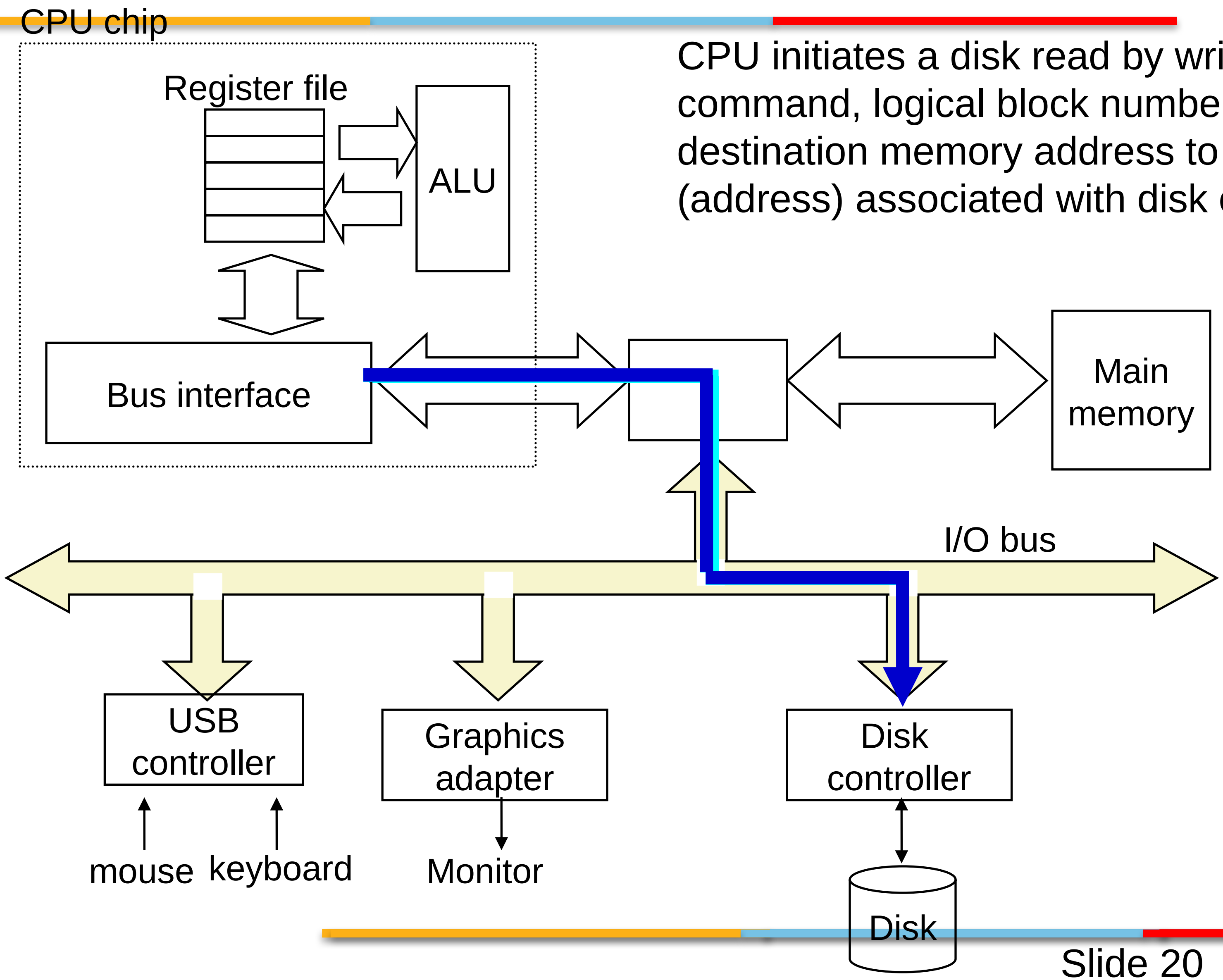
↑ Rotational latency

↑ Data transfer

I/O Bus

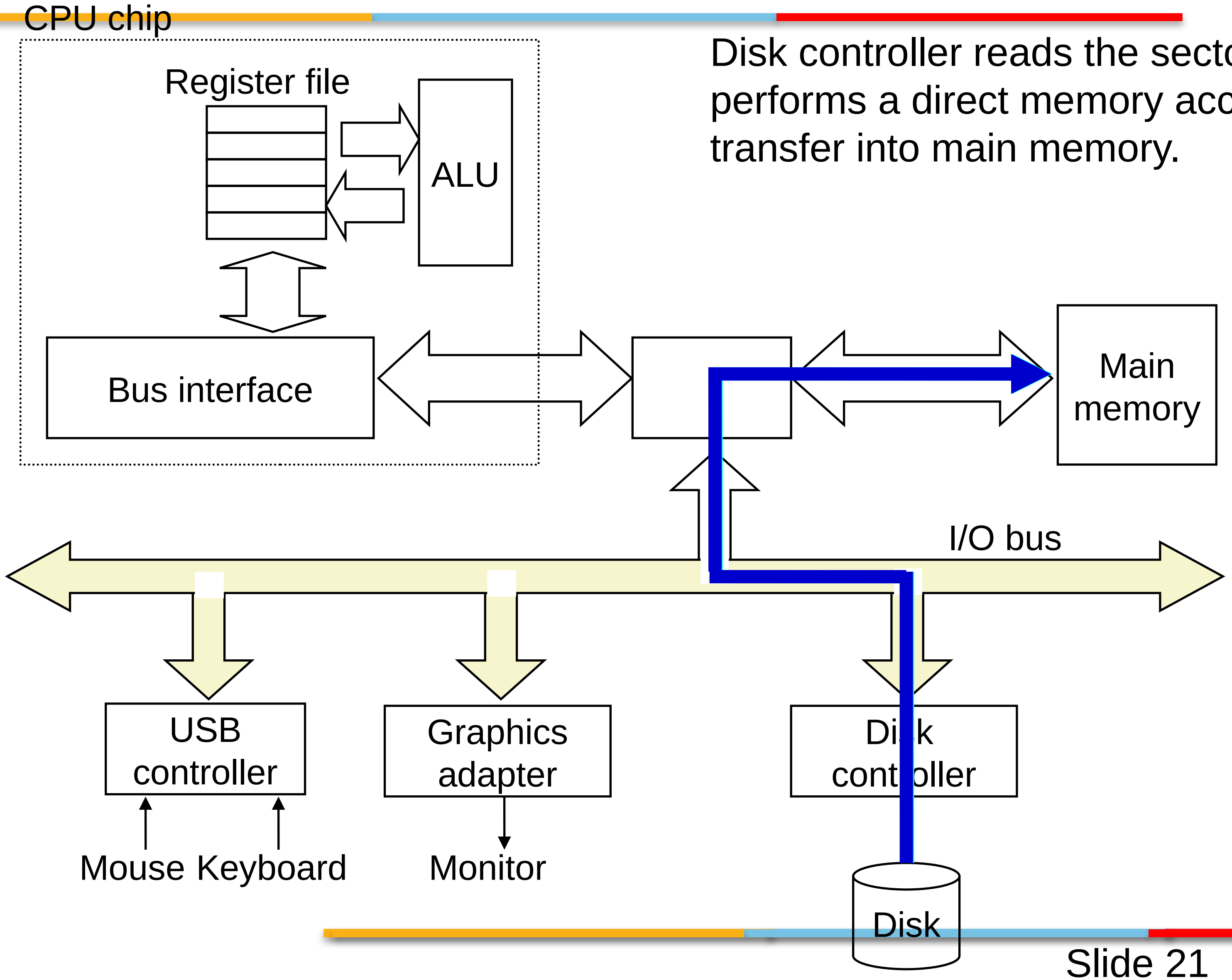

CPU chip
Register file
ALU
System bus
Memory bus
Bus interface
I/O bridge
Main memory
I/O bus
Expansion slots for other devices such as network adapters.
USB controller
Graphics adapter
Disk controller
Mouse Keyboard
Monitor
Disk

Reading a Disk Sector (1)


CPU chip
Register file
ALU
Bus interface
Main memory
I/O bus
USB controller
Graphics adapter
Disk controller
mouse
keyboard
Monitor
Disk

CPU initiates a disk read by writing a command, logical block number, and destination memory address to a port (address) associated with disk controller.

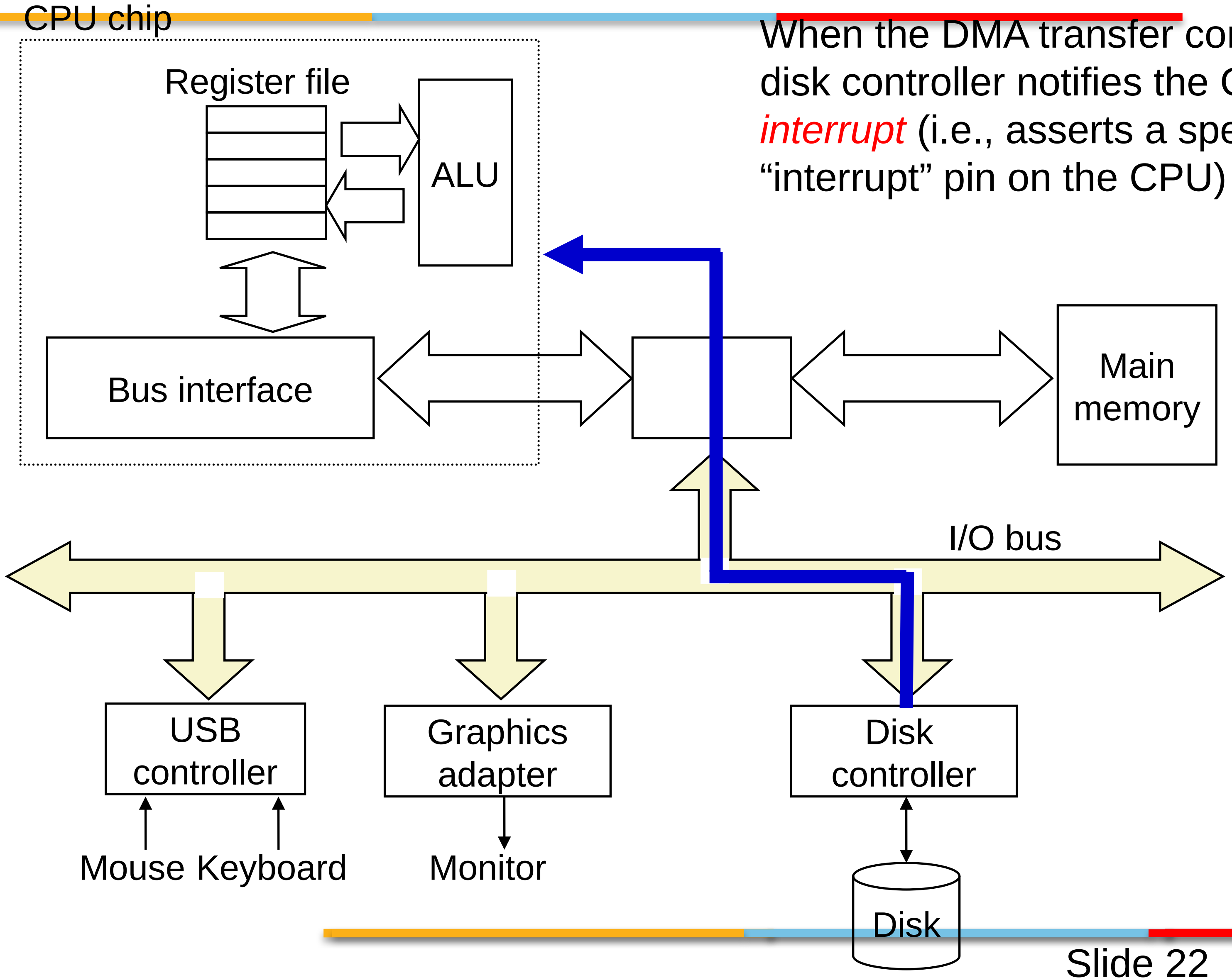
Reading a Disk Sector (2)


CPU chip
Register file
ALU
Bus interface
Main memory
I/O bus
USB controller
Graphics adapter
Disk controller
Mouse Keyboard
Monitor
Disk

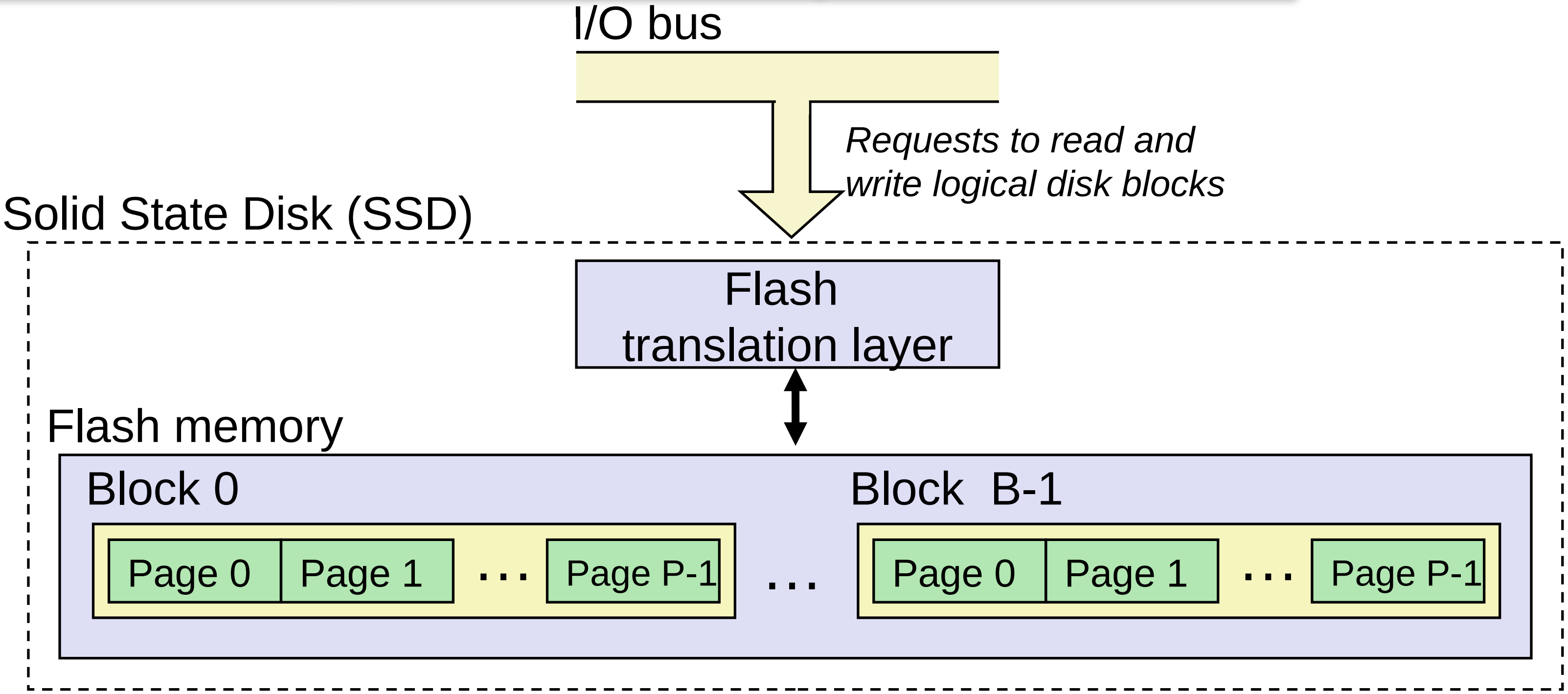
Disk controller reads the sector and performs a direct memory access (DMA) transfer into main memory.

Reading a Disk Sector (3)

When the DMA transfer completes, the disk controller notifies the CPU with an *interrupt* (i.e., asserts a special "interrupt" pin on the CPU)


CPU chip
Register file
ALU
Bus interface
Main memory
I/O bus
USB controller
Graphics adapter
Disk controller
Mouse Keyboard
Monitor
Disk

Solid State Disks (SSDs)



I/O bus

Requests to read and write logical disk blocks

Solid State Disk (SSD)

Flash translation layer

Flash memory

Block 0

Page 0 Page 1 … Page P-1

…

Block B-1

Page 0 Page 1 … Page P-1

- Pages: 512KB to 4KB, Blocks: 32 to 128 pages
- Data read/written in units of pages.
- Page can be written only after its block has been erased
- A block wears out after about 100,000 repeated writes.

SSD Performance Characteristics

Sequential read tput*	550 MB/s	Sequential write tput	470 MB/s
Random read tput	365 MB/s	Random write tput	303 MB/s
Avg seq read time	50 us	Avg seq write time	60 us

Tput → Throughput

- Sequential access faster than random access
  - Common theme in the memory hierarchy
- Random writes are somewhat slower
  - Erasing a block takes a long time (~1 ms)
  - Modifying a block page requires all other pages to be copied to new block
  - In earlier SSDs, the read/write gap was much larger.

Source: Intel SSD 730 product specification.

SSD Tradeoffs vs Rotating Disks

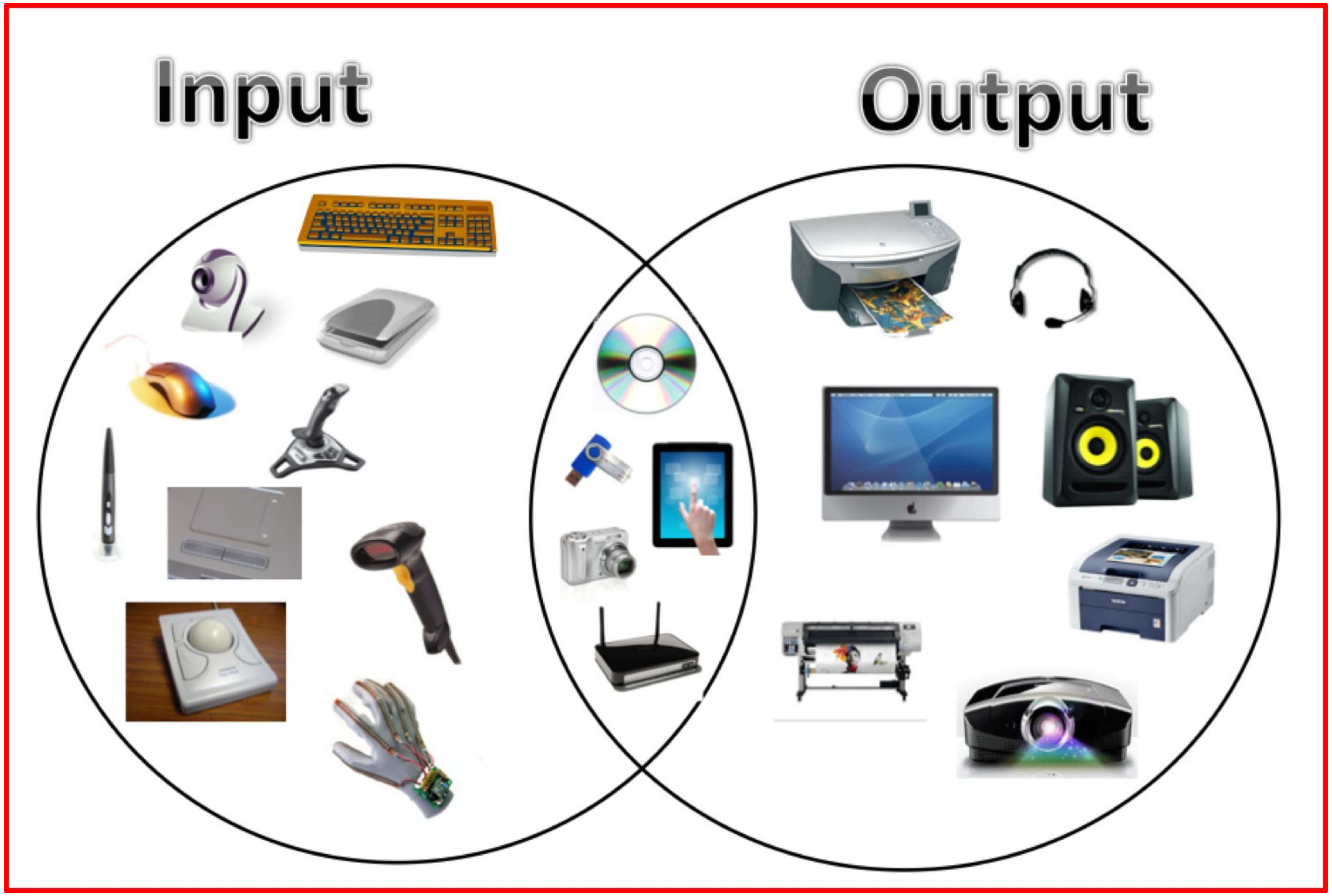
- Advantages
  - No moving parts ✉faster, less power, more rugged
- Disadvantages
  - Have the potential to wear out
    - Mitigated by “wear leveling logic” in flash translation layer
    - E.g. Intel SSD 730 guarantees 128 petabyte (128 x 10^{15} bytes) of writes before they wear out
  - In 2015, about 30 times more expensive per byte
- Applications
  - MP3 players, smart phones, laptops
  - Beginning to appear in desktops and servers

I/O Abstraction (Contd…)

Today’s topic

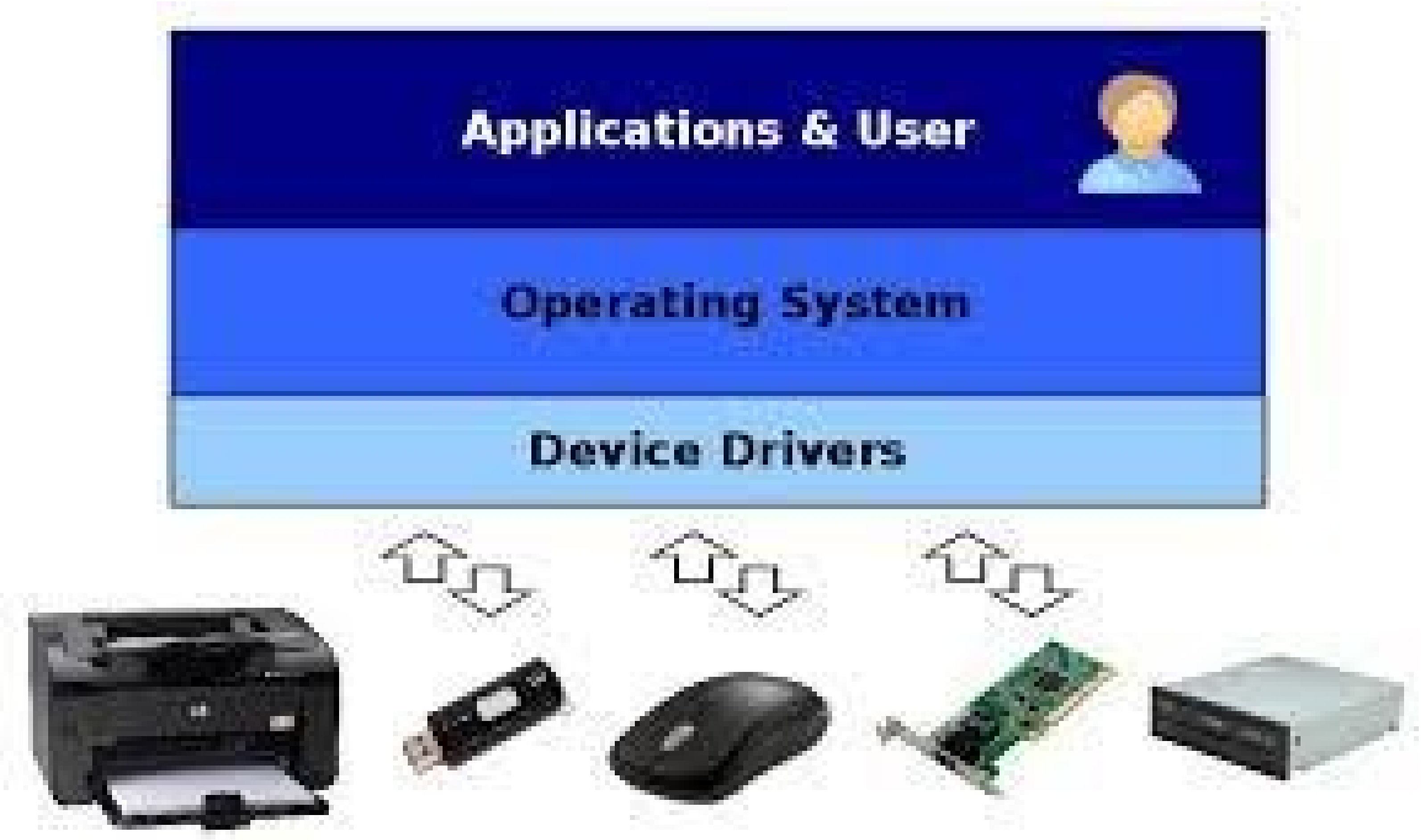
- **I/O System Software**
  - Device Drivers
  - Device Driver Interfaces
    - The Block Device Interfaces
    - The Character Device Interfaces
  - Device Numbers
- **Unification of Files and I/O Devices**
- **Mechanisms for improving I/O performance**
  - Prefetching
  - Caching
  - Buffering

Input and output devices


Input
Output

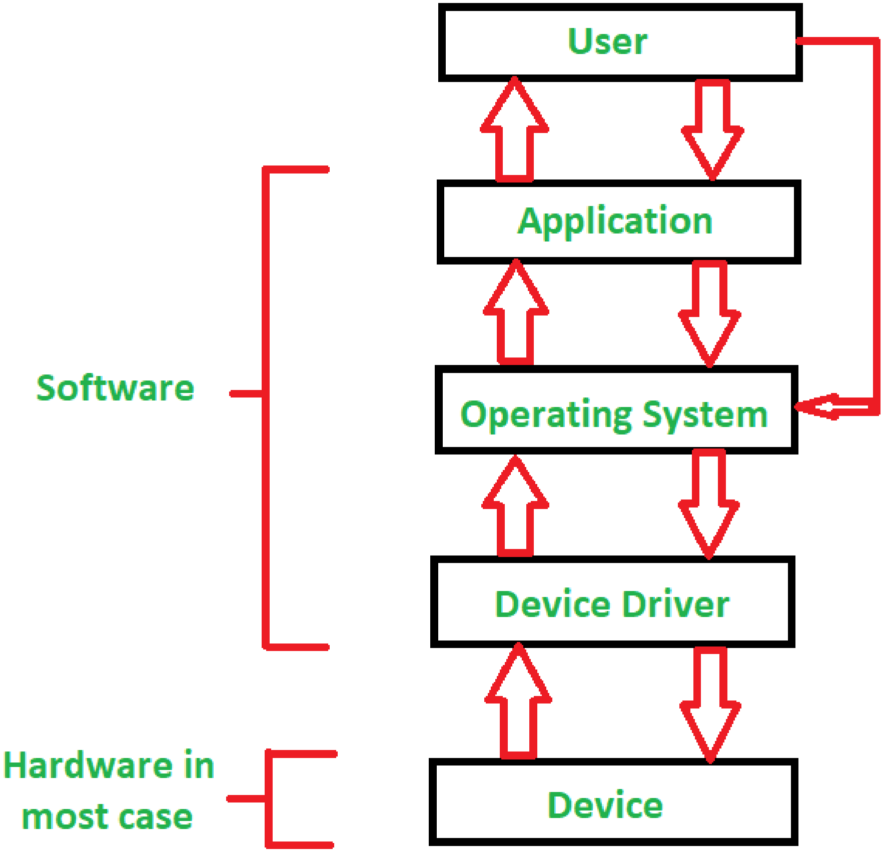
DEVICE DRIVERS

Software that allows your computer to communicate to hardware devices


Applications & User
Operating System
Device Drivers

DEVICE DRIVERS

Abstract layer or translation layer



User
Application
Operating System
Device Driver
Device
Software
Hardware in most case

DEVICE DRIVERS

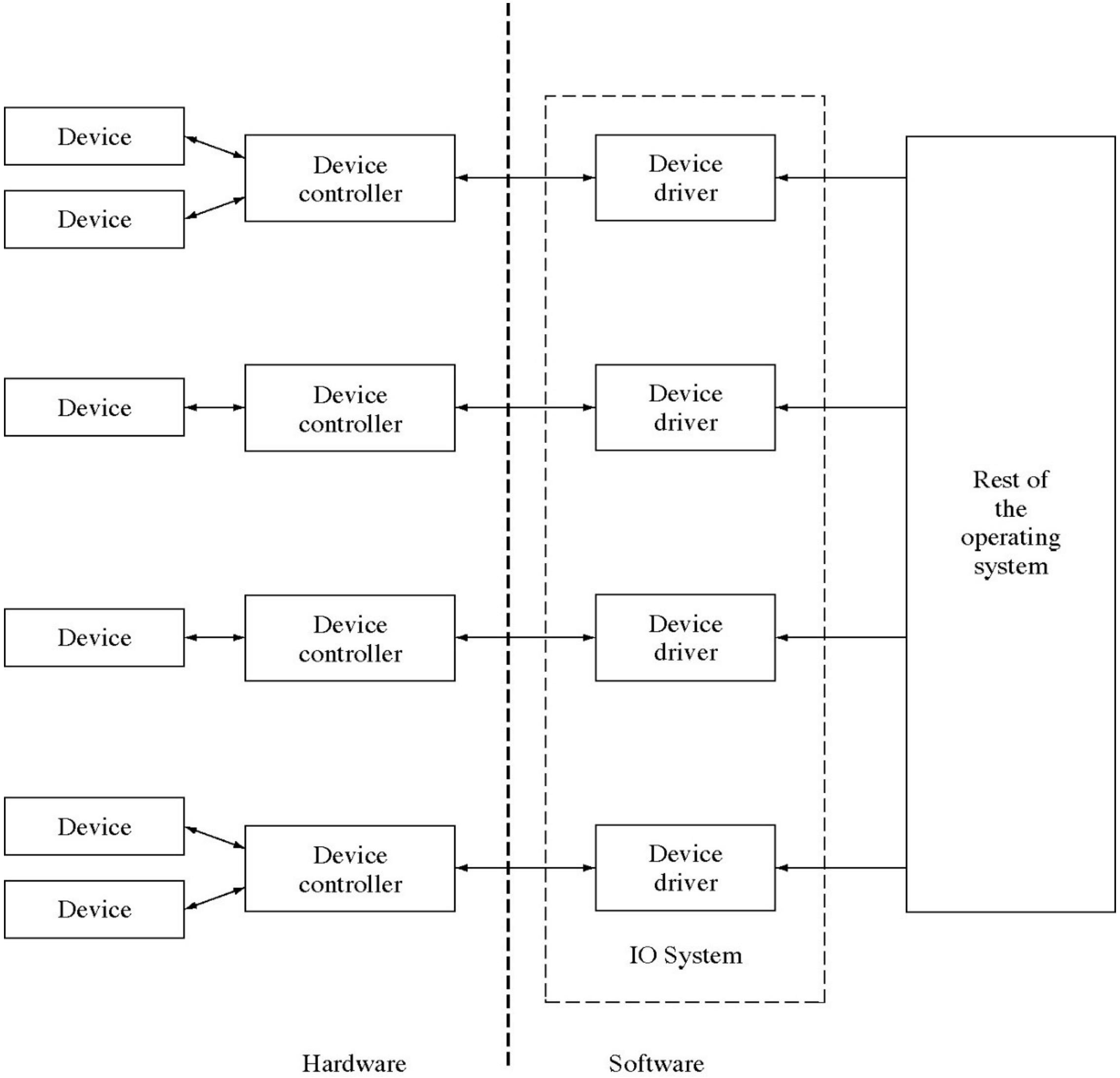
- **Devices** – Complex and hard to use

- It is controlled by Device Controllers through Device Controller register (SCSI Devices)

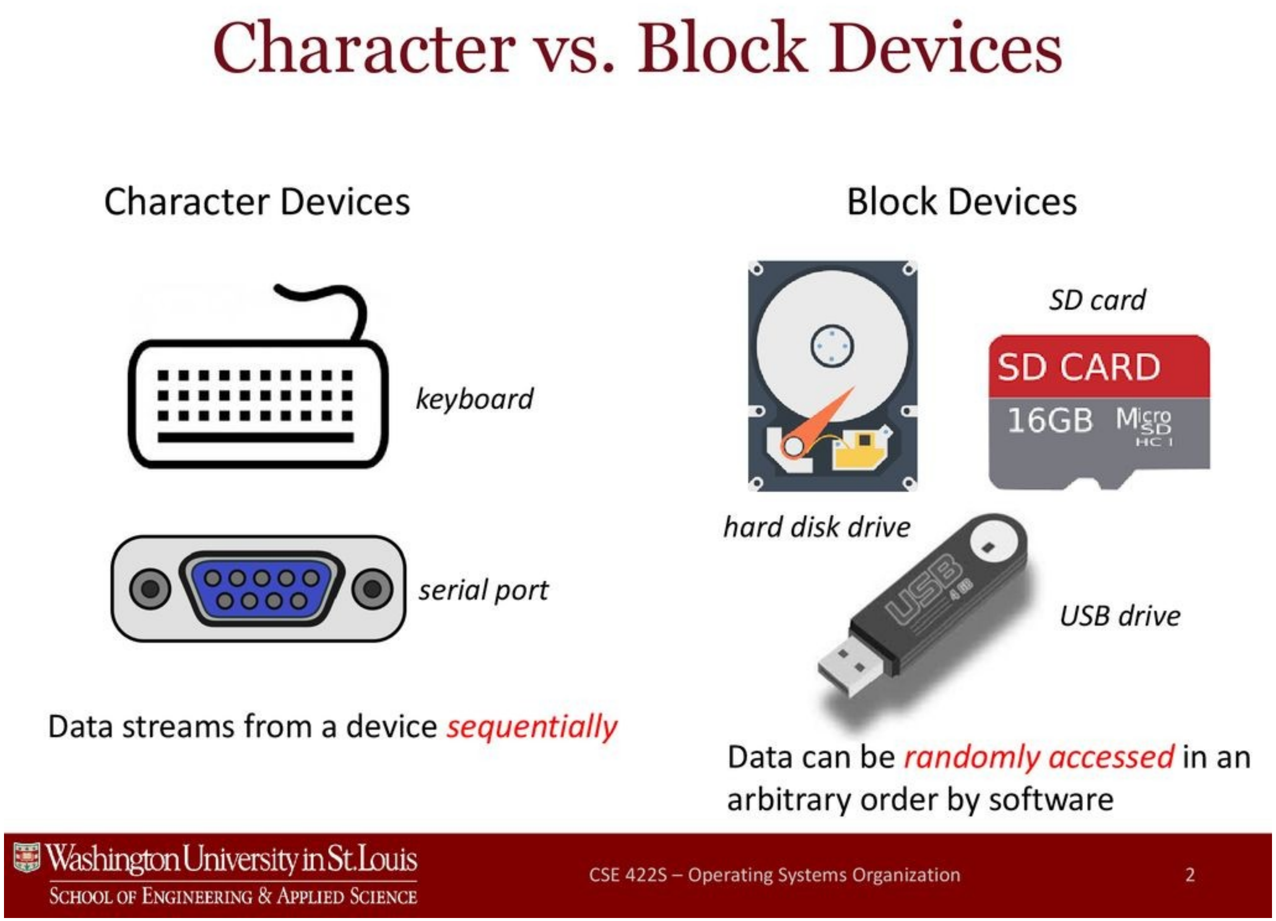
- **Device Driver** – the modules that contains everything about device controller and the device attached to it.

- An interface module that communicates with both device language and OS language.

Device Drivers in an Operating System


Device
Device
Device controller
Device driver
Device
Device controller
Device driver
Device
Device controller
Device driver
Device
Device
Device controller
Device driver
IO System
Rest of the operating system
Hardware
Software

The Two Categories of Device Drivers



Character vs. Block Devices

Character Devices

keyboard
serial port

Data streams from a device *sequentially*

Block Devices

SD card
SD CARD
16GB
Micro SD
HC I
hard disk drive
USB
4 GB
USB drive

Data can be *randomly accessed* in an arbitrary order by software

The Two Categories of Device Drivers

- **Block Device Interface –** the device drivers will all have the same interface; **Block Devices** – A device with fixed size, addressable blocks

- **Character Device (Character Device interface)** – a non-block devices send or receive data in streams of bytes with no addresses associated with the bytes and with no fixed block size.

- Both these interfaces have an open and close.

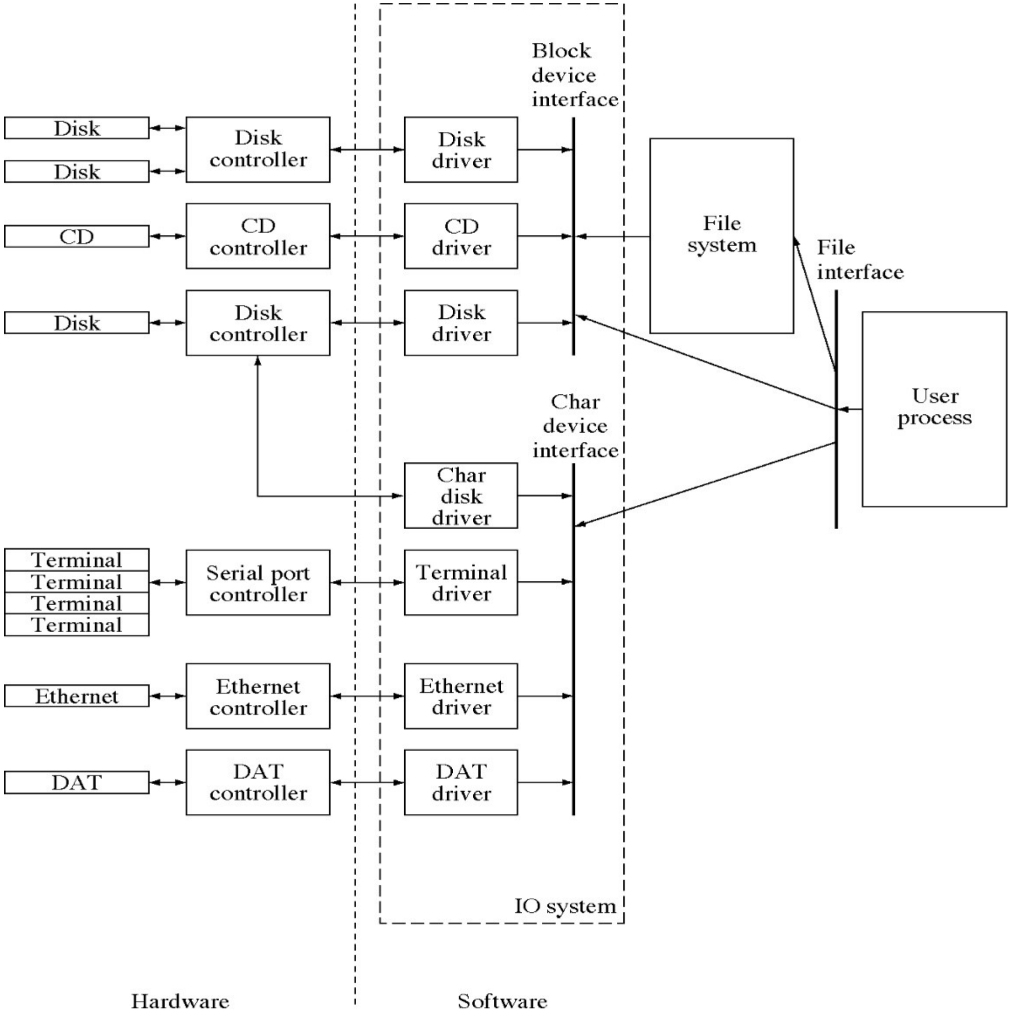
DEVICE NUMBER

- The main use of device numbers is to name which device to use (among those controlled by a single controller)
- But they are often used to convey other information, for example:
  - device 0 rewinds when done, device 8 does not
  - device 0 uses normal-sized paper, device 8 uses legal-sized paper
  - device 0 writes at high density, device 1 at medium density and device 2 at low density

UNIFICATION OF FILES AND I/O DEVICES

- The device driver interface is nearly the same as the file interface
- The file interface is a set of operation – open, read, write, seek and close
- The I/O system has code to translate from the file interface to the device driver interface, so file commands can be used on devices

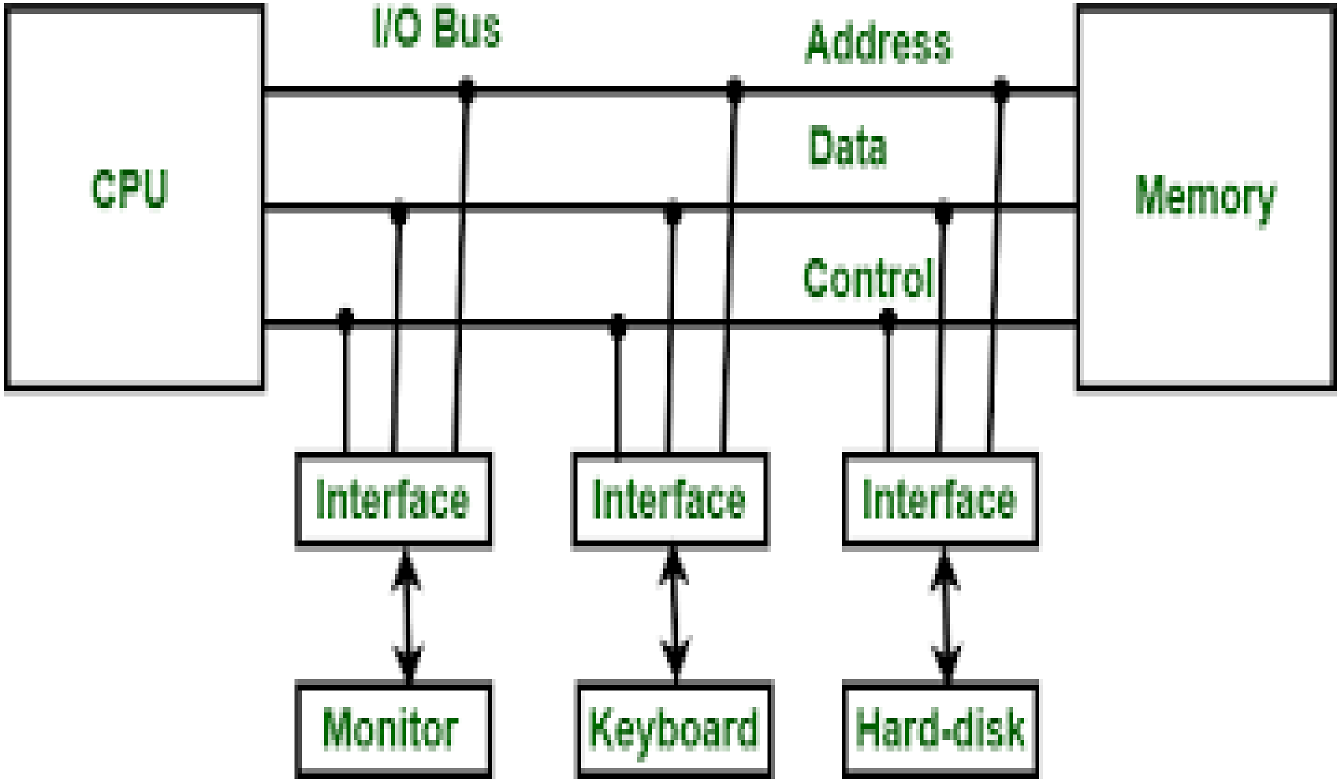
The File, Character, and Block Device Interfaces


Block device interface
Disk
Disk
Disk controller
Disk driver
CD
CD controller
CD driver
Disk
Disk controller
Disk driver
File system
File interface
User process
Char device interface
Char disk driver
Terminal
Terminal
Terminal
Terminal
Serial port controller
Terminal driver
Ethernet
Ethernet controller
Ethernet driver
DAT
DAT controller
DAT driver
IO system
Hardware
Software

The File, Character, and Block Device Interfaces

- There are block devices that are connected to device controllers and managed by device drivers.
- All block device drivers have same block device interface.
- All character device drivers have same character device interface
- The file system uses the block device interface and exports the file interface for files.
- The I/O system uses block device interface and character device interface and exports the file interface for devices.
- The user only sees the file interface for both files and devices.

The File, Character, and Block Device Interfaces

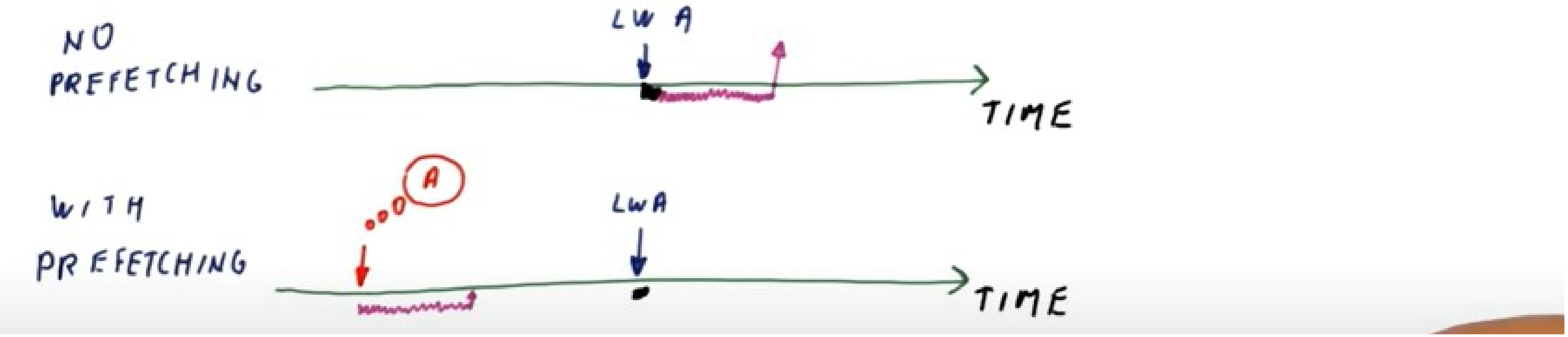

I/O Bus
Address
Data
Control
CPU
Memory
Interface
Interface
Interface
Monitor
Keyboard
Hard-disk

Mechanisms for Improving I/O Performance - Prefetching

- Prefetching is a technique for speeding up fetch operations by beginning a fetch operation whose result is expected to be needed soon.
- Prefetching allows a browser to silently fetch the necessary resources needed to display content that a user might access in the near future.
- The browser stores these resources in its cache enabling it to deliver the requested data faster.

Mechanisms for Improving I/O Performance

- Prefetching

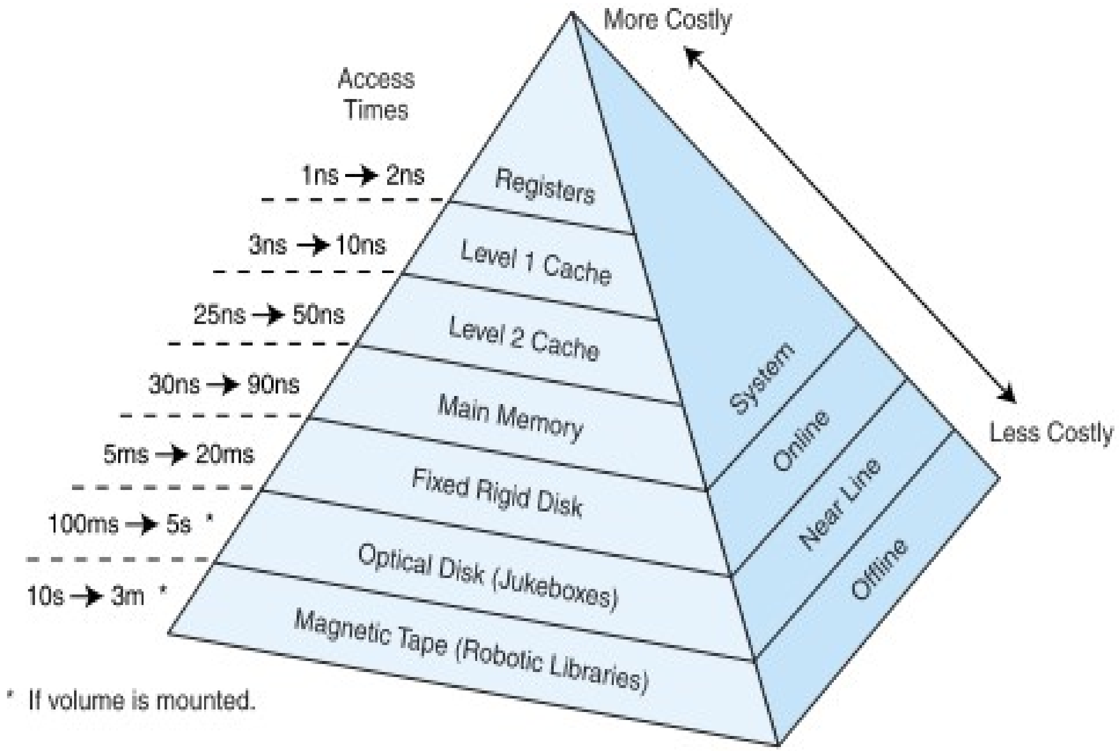

NO PREFETCHING
LW A
TIME
WITH PREFETCHING
A
LWA
TIME

1. Stream buffer
2. Stride pre-fetcher
3. Correlating pre-fetcher

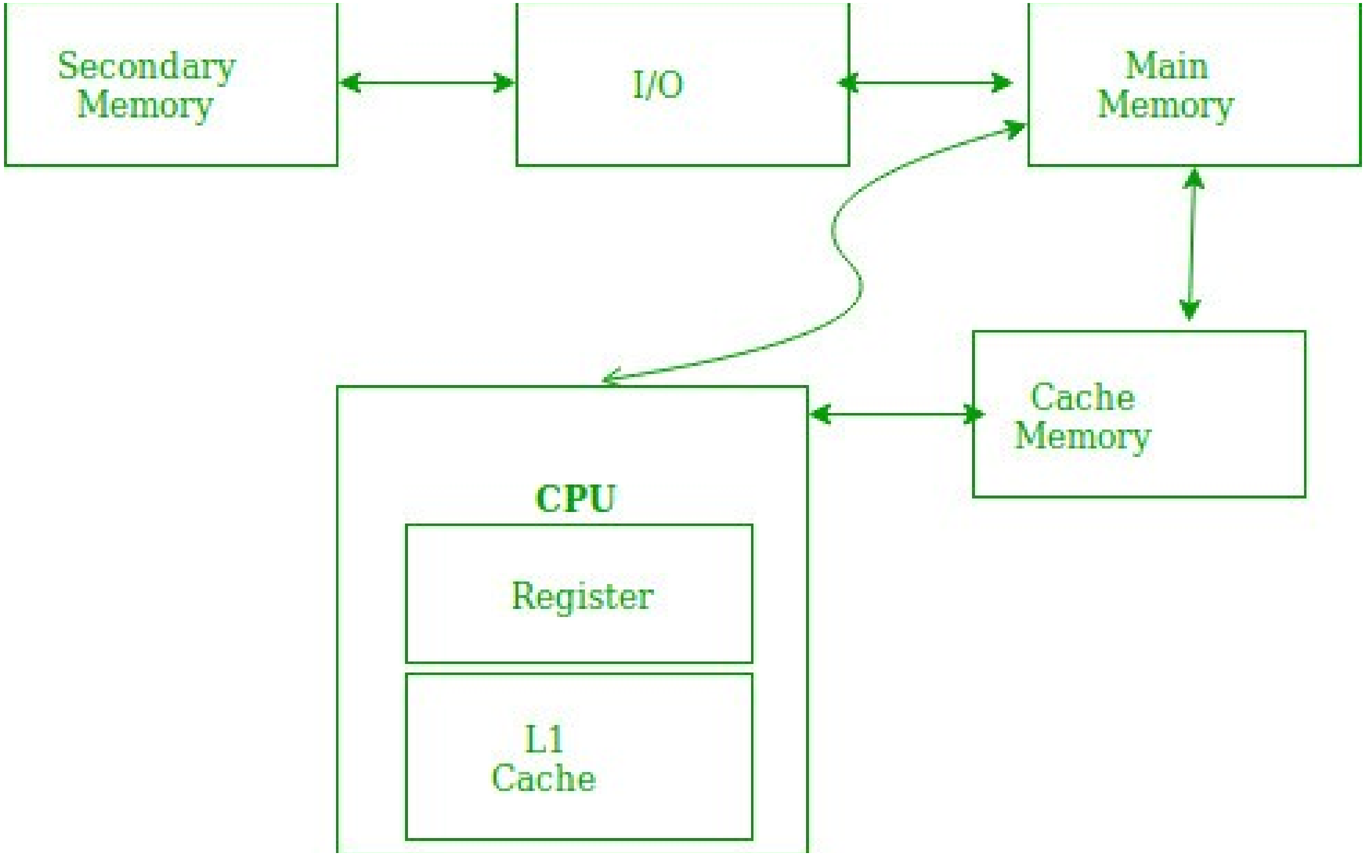
Caching

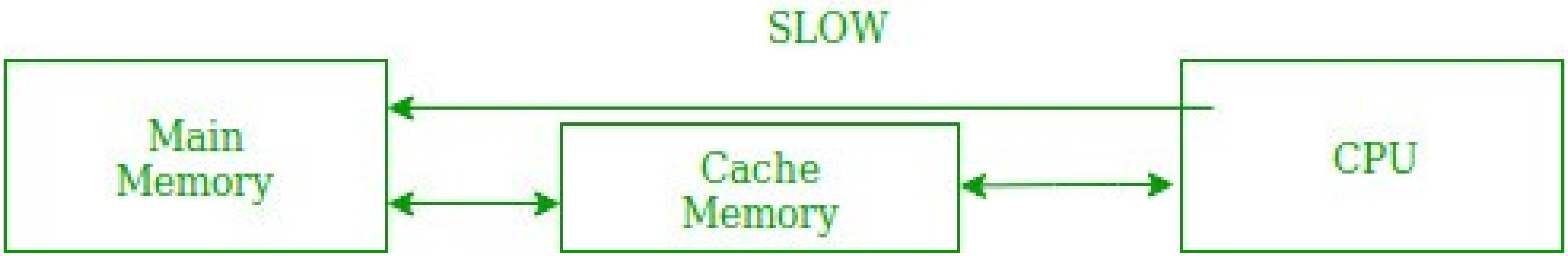
- **Caching -** storing data in a separate disk (very fast speed disk).
- The data which is to be used many times results in wastage of time if it is in hard disk but storing the data in cache reduces this time wastage.
- Cache is in processor and can be also implemented with RAM and disk.
- Read and write is same as the normal storage
- **Example –** Cache is used in system to speed up the access of data frequently used.

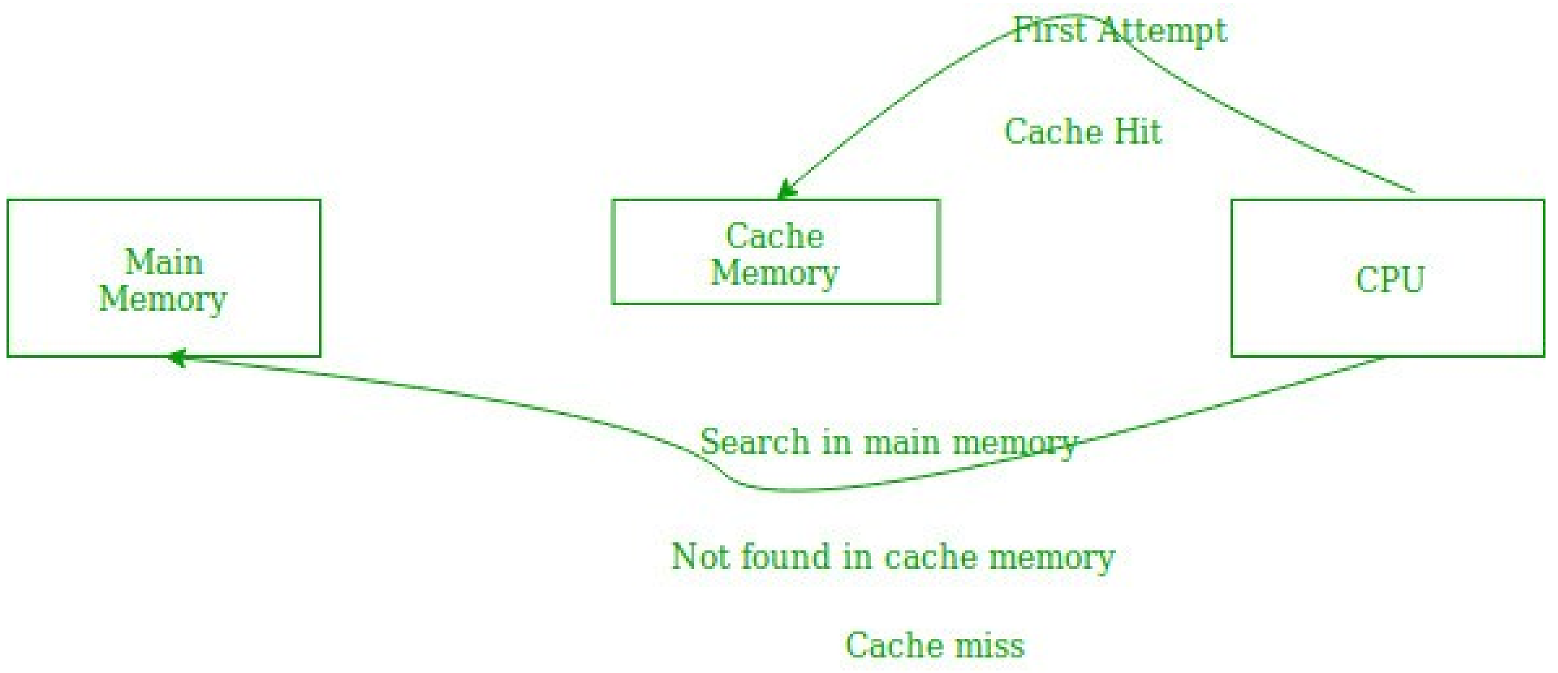
Computer Memory Hierarchy


More Costly
Access Times
1ns → 2ns
Registers
3ns → 10ns
Level 1 Cache
25ns → 50ns
Level 2 Cache
30ns → 90ns
Main Memory
5ms → 20ms
Fixed Rigid Disk
100ms → 5s *
Optical Disk (Jukeboxes)
10s → 3m *
Magnetic Tape (Robotic Libraries)
System
Online
Near Line
Offline
Less Costly
* If volume is mounted.

Source: Null, Linda and Lobur, Julia (2003). *Computer Organization and Architecture* (p. 236). Sudbury, MA: Jones and Bartlett Publishers


Secondary Memory
I/O
Main Memory
Cache Memory
CPU
Register
L1 Cache


SLOW
Main Memory
Cache Memory
CPU


First Attempt
Cache Hit
Main Memory
Cache Memory
CPU
Search in main memory
Not found in cache memory
Cache miss

Principle of Locality

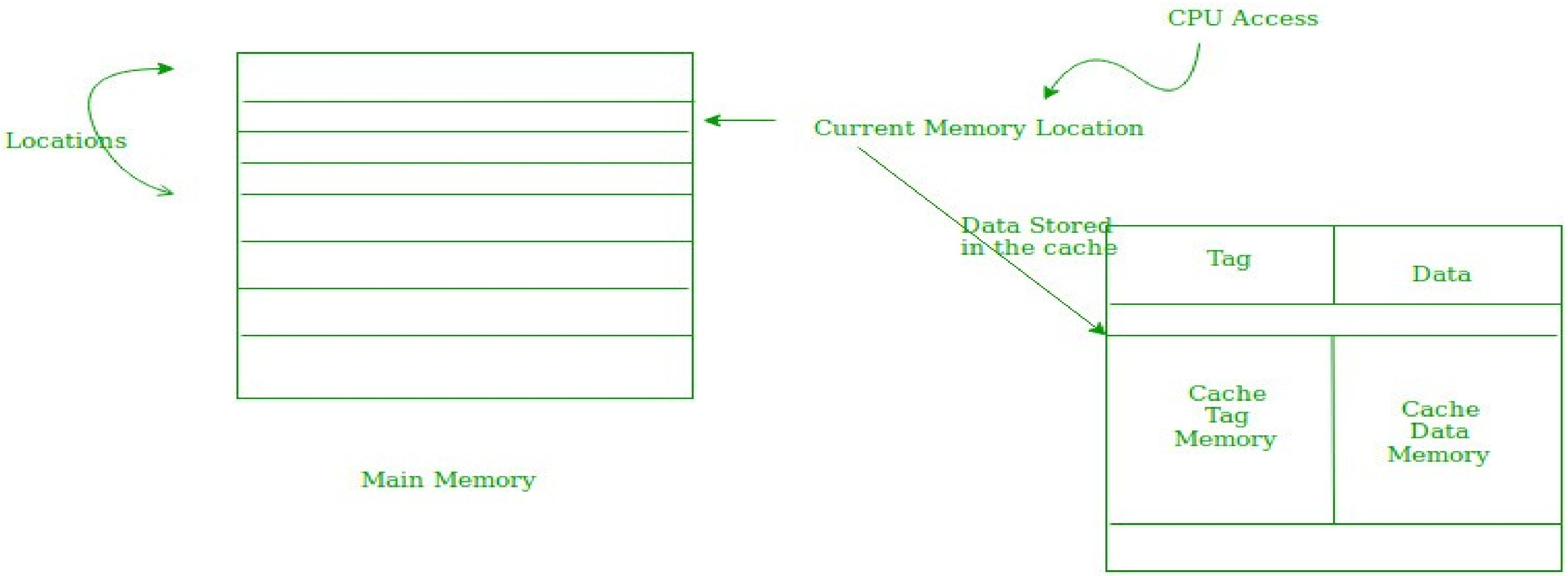
locality of reference, also known as the **principle of locality**, is the tendency of a processor to access the same set of memory locations repetitively over a short period of time.

- .

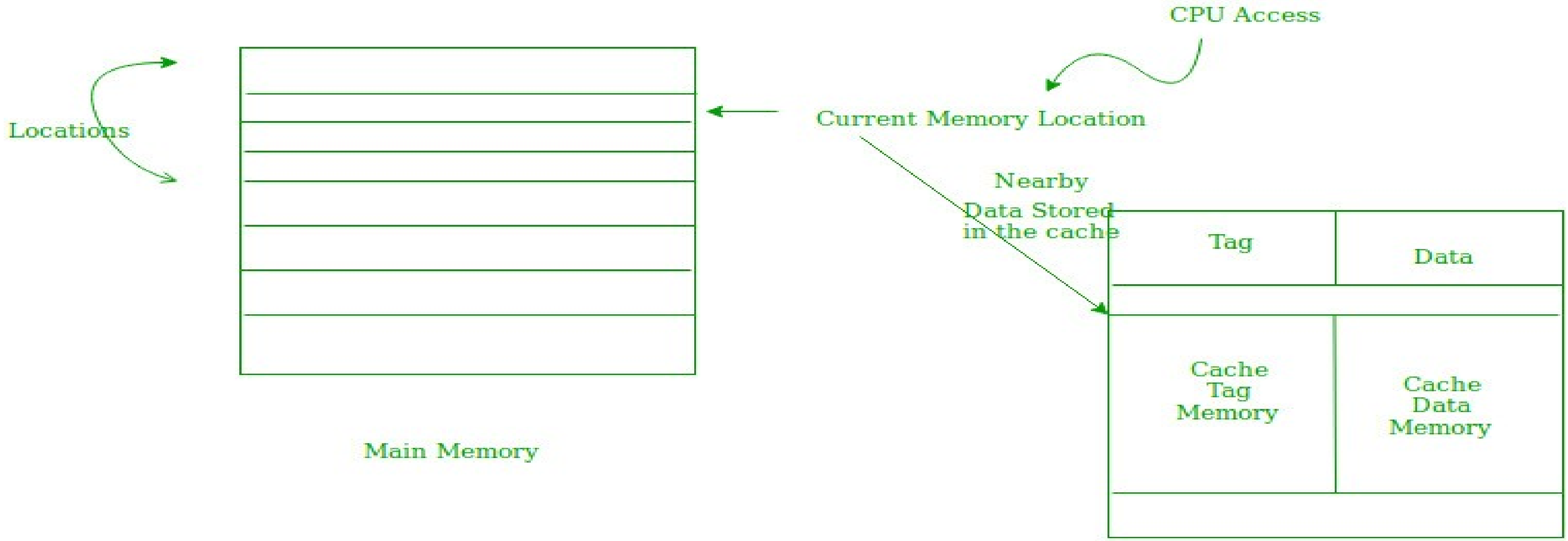
In case of loops in program control processing unit repeatedly refers to the set of instructions that constitute the loop.

In case of subroutine calls, everytime the set of instructions are fetched from memory.

References to data items also get localized that means same data item is referenced again and again


CPU Access
Locations
Current Memory Location
Data Stored in the cache
Tag
Data
Cache Tag Memory
Cache Data Memory
Main Memory

- **Temporal locality means current data or instruction that is being fetched may be needed soon. So we should store that data or instruction in the cache memory so that we can avoid again searching in main memory for the same data**


CPU Access
Locations
Current Memory Location
Nearby
Data Stored
in the cache
Tag
Data
Cache
Tag
Memory
Cache
Data
Memory
Main Memory

Spatial locality means instruction or data near to the current memory location that is being fetched, may be needed soon in the near future. This is slightly different from the temporal locality. Here we are talking about nearly located memory locations while in temporal locality we were talking about the actual memory location that was being fetched.

Cache Performance

1. The performance of the cache is measured in terms of hit ratio.
2. When CPU refers to memory and find the data or instruction within the Cache Memory, it is known as cache hit.
3. If the desired data or instruction is not found in the cache memory and CPU refers to the main memory to find that data or instruction, it is known as a cache miss.

```
Hit + Miss = Total CPU
Reference
Hit Ratio(h) = Hit /
(Hit+Miss)
```

- Average access time of any memory system consists of two levels: Cache and Main Memory. If *Tc* is time to access cache memory and *Tm* is the time to access main memory then we can write:

```
Tavg = Average time to access memory
Tavg = h * Tc + (1-h)*(Tm + Tc)
```

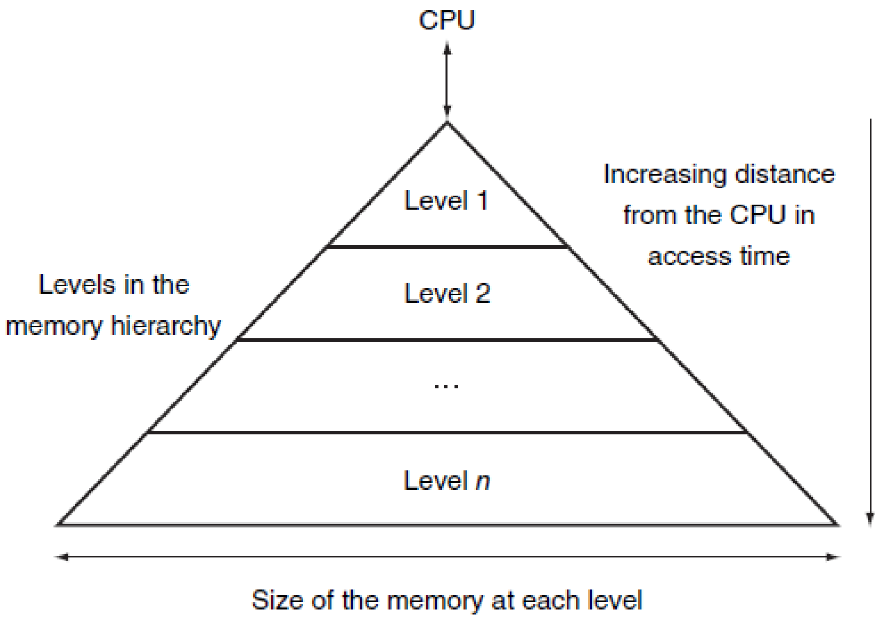
Memory Hierarchy..

- A memory hierarchy consists of multiple levels of memory with different speeds and sizes.
- The faster memories are more expensive per bit than the slower memories and thus are smaller.
- The goal is to present the user with as much memory as is available in the cheapest technology, while providing access at the speed offered by the fastest memory
- A memory hierarchy can consist of multiple levels, but data is copied between only two adjacent levels at a time

- The upper level—the one closer to the processor—is smaller and faster than the lower level, since the upper level uses technology that is more expensive
- If the data requested by the processor appears in some block in the upper level, this is called a ***hit***
- If the data is not found in the upper level, the request is called a ***miss*.**

- **block (or line)** The minimum unit of information that can be either present or not present in a cache.
- **hit rate** The fraction of memory accesses found in a level of the memory hierarchy.(Also called hit ratio)
- **miss rate** The fraction of memory accesses not found in a level of the memory hierarchy (1-hit rate)
- **miss penalty** The time required to fetch a block into a level of the memory hierarchy from the lower level, including the time to access the block, transmit it from one level to the other, insert it in the level that experienced the miss, and then pass the block to the requestor.

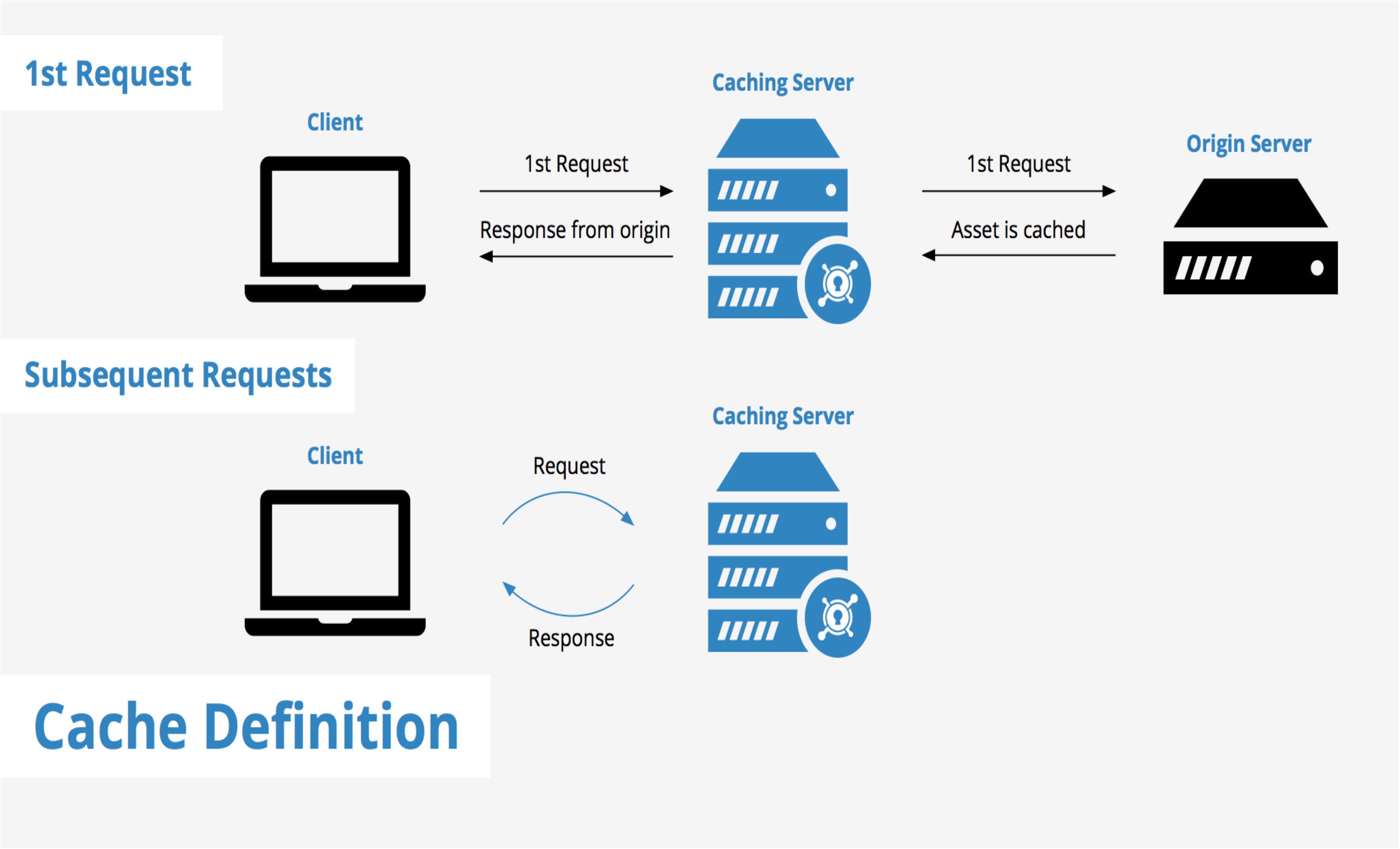
Memory Hierarchy..


CPU
Level 1
Level 2
...
Level n
Levels in the memory hierarchy
Increasing distance from the CPU in access time
Size of the memory at each level

Structure of Memory Hierarchy: As the distance from processor increases, so does the size

Cache Memory

A cache is a smaller, faster memory, located closer to a processor core, which stores copies of the data from frequently used main memory locations.


1st Request
Client
Caching Server
Origin Server
1st Request
Response from origin
1st Request
Asset is cached
Subsequent Requests
Client
Caching Server
Request
Response

Cache Definition

Buffering

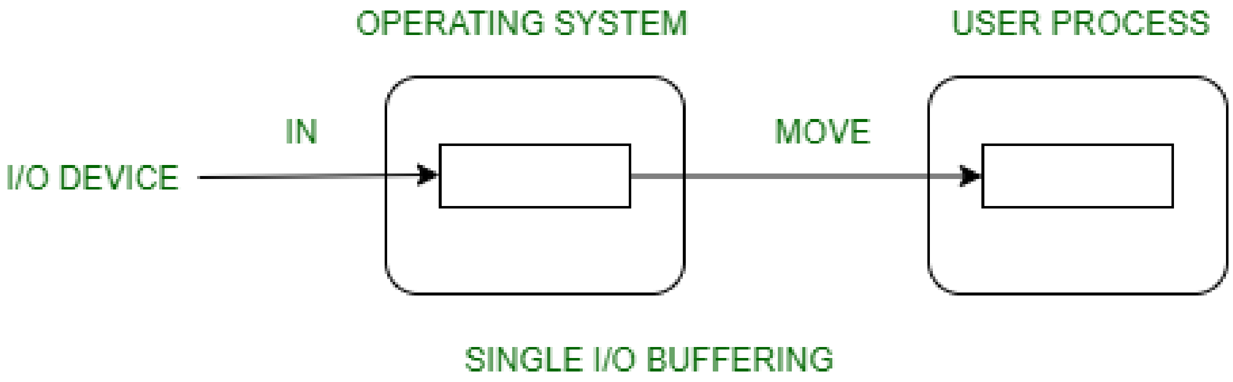
- A buffer is a memory area that stores data being transferred between two devices or between a device and an application
- Buffering is done to deal effectively with a speed mismatch between the producer and consumer of the data stream.
- This process of data transfer is not instantaneous, therefore it needs another buffer in order to store additional incoming data.
- Buffering also provides variations for devices that have different data transfer sizes.

Uses of I/O Buffering

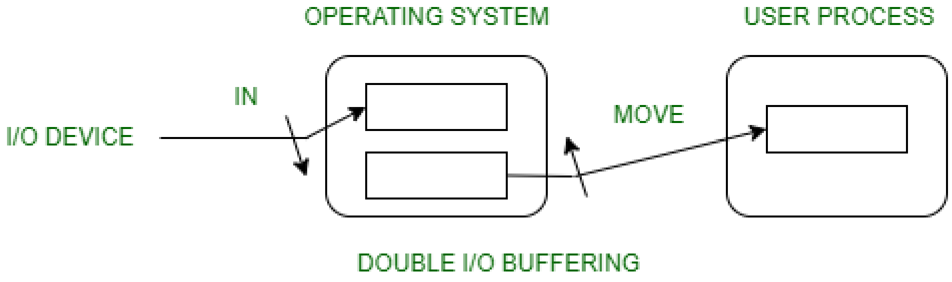
- A buffer is produced in main memory to heap up the bytes received from modem.
- Provides variations for devices that have different data transfer sizes.
- The use of two buffers disintegrates the producer and the consumer of the data, thus minimizes the time requirements between them.

Types of I/O Buffering Techniques

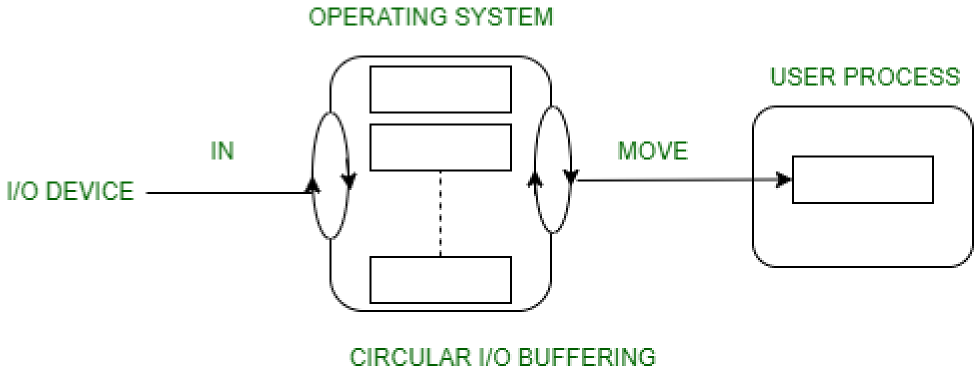
- **Single buffer -** A buffer is provided by the operating system to the system portion of the main memory.


OPERATING SYSTEM
USER PROCESS
IN
MOVE
I/O DEVICE
SINGLE I/O BUFFERING

- **Double buffer**


OPERATING SYSTEM
USER PROCESS
IN
I/O DEVICE
MOVE
DOUBLE I/O BUFFERING

- **Circular buffer -** When more than two buffers are used, the collection of buffers is itself referred to as a circular buffer. (producer-consumer)


OPERATING SYSTEM
USER PROCESS
IN
MOVE
I/O DEVICE
CIRCULAR I/O BUFFERING

THANK YOU

BITS Pilani
Pilani Campus

BITS Pilani
Pilani Campus

COMPUTING AND DESIGN

SESSION 11

Course design by - Dr. Lucy J. Gudino &
Dr Chandrasekhar
Faculty – Dr Thangakumar J
WILP & Department of CS & IS

BITS Pilani
Pilani Campus

Operating Systems – File as a Logical Abstraction of Storage

Last Session

List of Topic Title	Text/Ref Book/external resource
• **I/O System Software** o Device Drivers o Device Driver Interfaces ▪ The Block Device Interfaces ▪ The Character Device Interfaces o Device Numbers • **Unification of Files and I/O Devices** • **Mechanisms for improving I/O performance** o Prefetching o Caching o Buffering	T1 (15.1, 15.4) T1 (15.5)

File as a Logical Abstraction of Storage

Today’s topic

- Levels in the file system
- File storage on hard disks, CD ROMs and Flash ROMs
- File Vs Databases

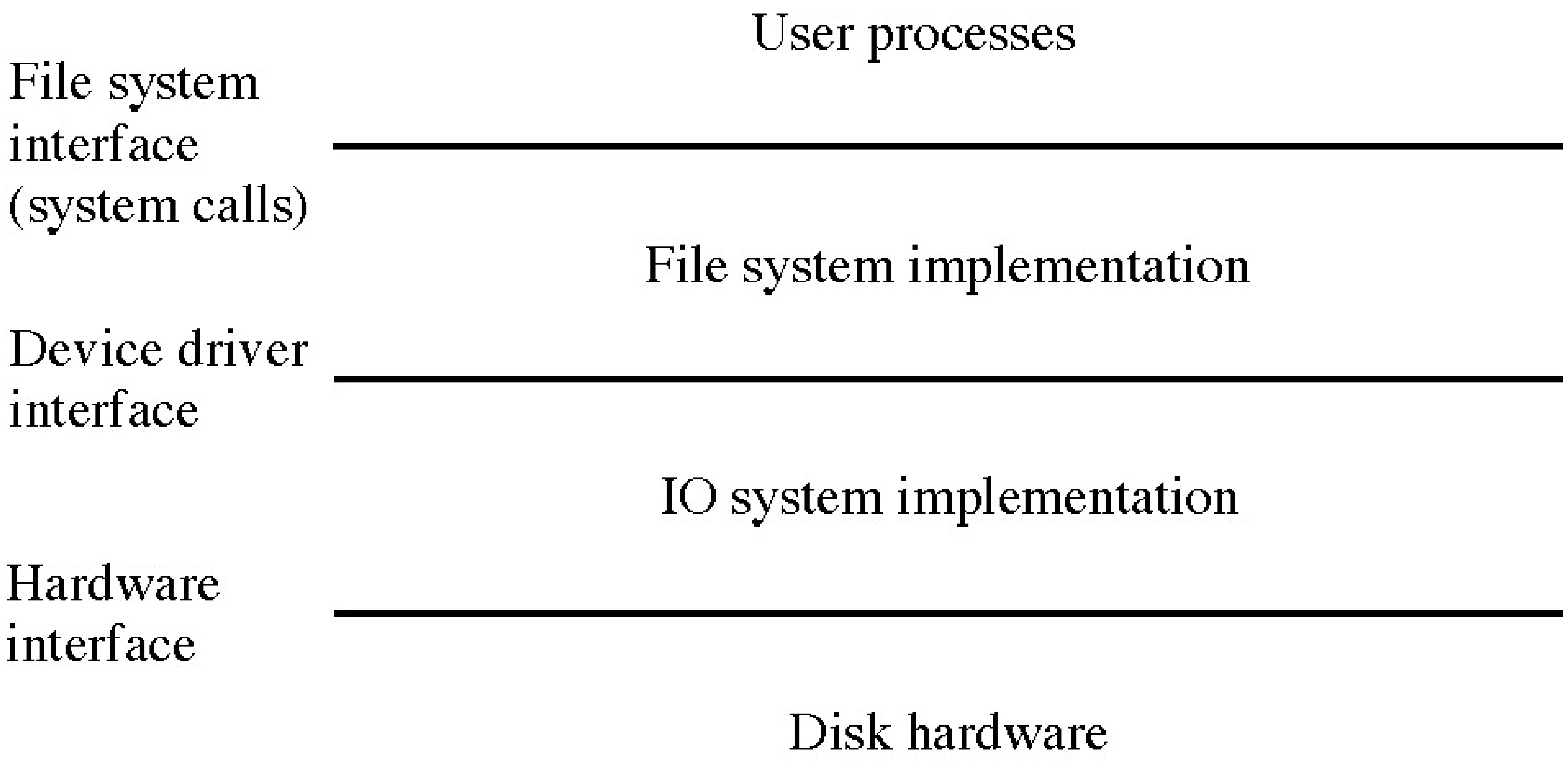
FILES

- A **File** – a sequence of bytes of arbitrary length (0 to large upper limit)

- It has name (a character string) which shows the fits into a hierarchical structure.

- Implemented by the OS to provide persistent storage.

LEVELS IN THE FILE SYSTEM



User processes

File system interface (system calls)

File system implementation

Device driver interface

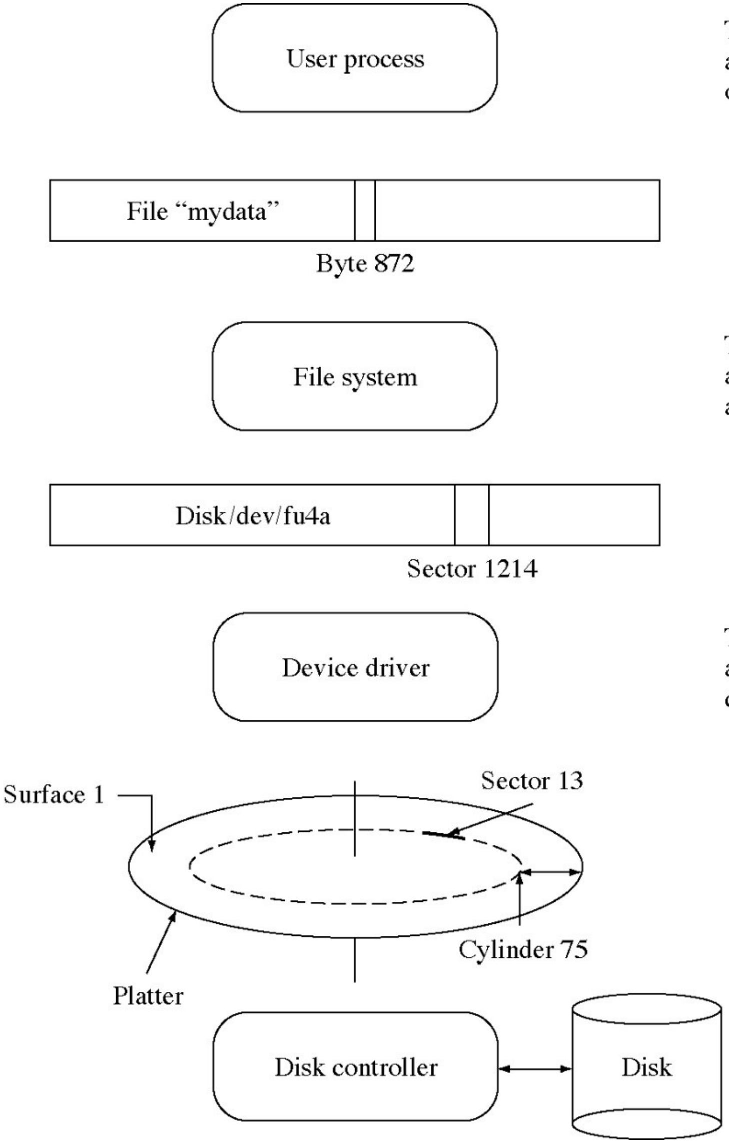
IO system implementation

Hardware interface

Disk hardware

- Figure shows the levels of file system abstraction.
- Logical specification of the system call interface to the file system objects and operations.
- Implementation of the file system interface on the I/O system interface
- Implementation of the I/O system interface on the disk hardware

Views of a File at Each Level


User process
File "mydata"
Byte 872
File system
Disk/dev/fu4a
Sector 1214
Device driver
Surface 1
Sector 13
Cylinder 75
Platter
Disk controller
Disk

The user process sees a file that is an array of bytes

The file system sees a logical disk that is an array of sectors

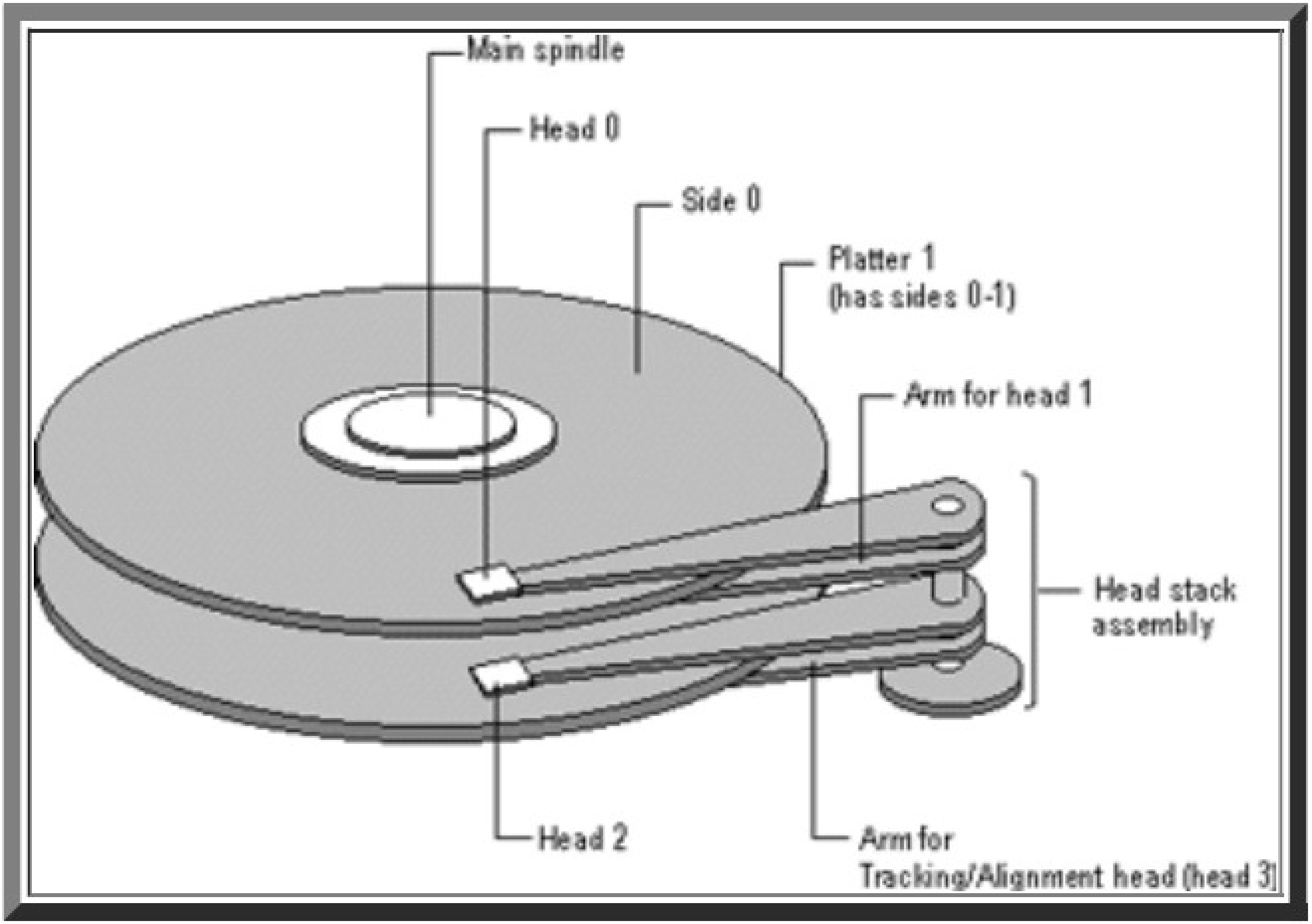
The disk driver sees a disk of platters, cylinders, and sectors

- Figure shows the views of files at each of the levels of file system abstraction
- Many possible variations in how the file abstraction can be presented to the user processes
- Many important design issues in file system interface design

Hard Disk Drive (HDD)

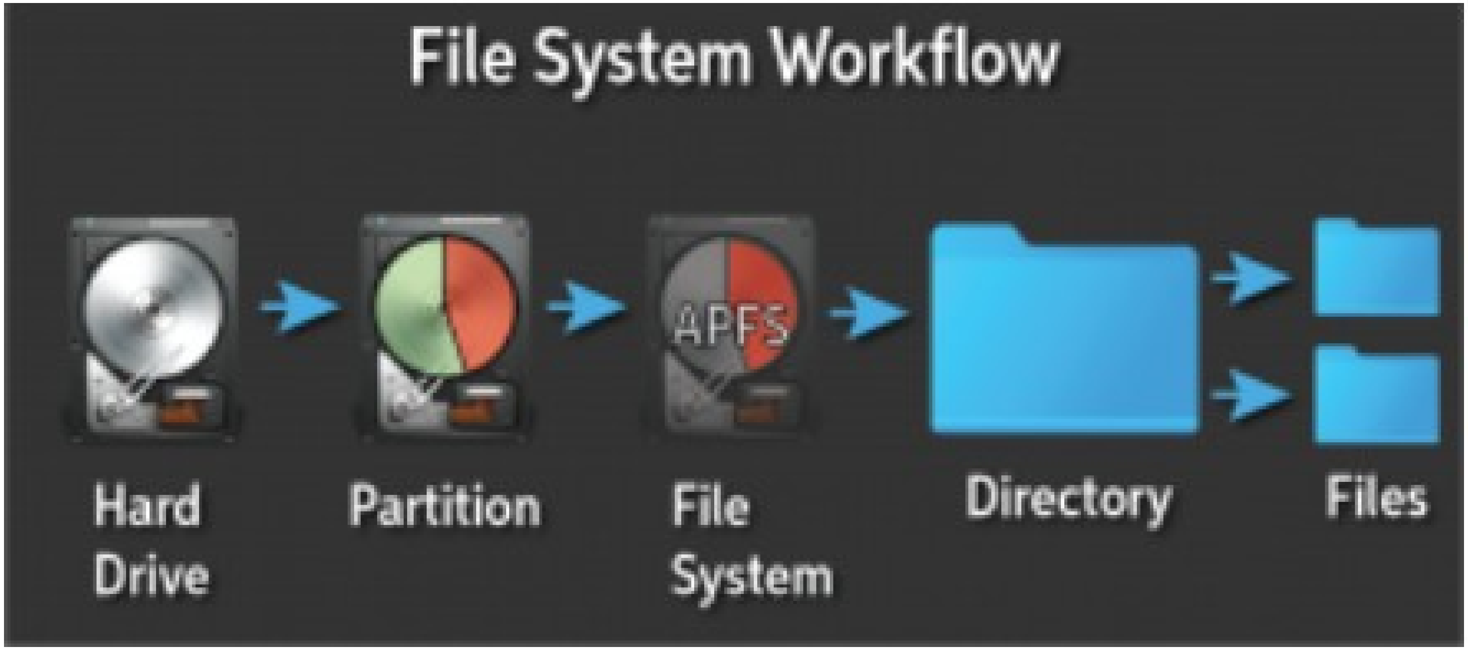
- A non-volatile data storage device
- Usually installed inside desktop computers, mobile devices, consumer electronics
- Used either as the primary or secondary storage device
- External hard drives can be used to expand the storage capacity of a computer or to act as a portable device to back up data.
- Users can store data from multiple devices and physically bring that data with them wherever they go.

Structure of a Hard Disk


Main spindle
Head 0
Side 0
Platter 1
(has sides 0-1)
Arm for head 1
Head stack
assembly
Head 2
Arm for
Tracking/Alignment head (head 3)

- Hard disk consists of a number of platters
- The platters rotate at a very high speed (5400 RPM to 10,000 RPM)
- Disk (read/write) heads move over the platter surface to read and write (magnetize) data bits
- The disk head can read or write data only when the desired disk surface area is under the disk head.
- Each platter is logically divided into a number of tracks
- Each track is logically divided into a number of sectors
- Each sector is uniquely identified by its sector number
- Disk sectors are usually 512 bytes in size.

Workflow of a File System

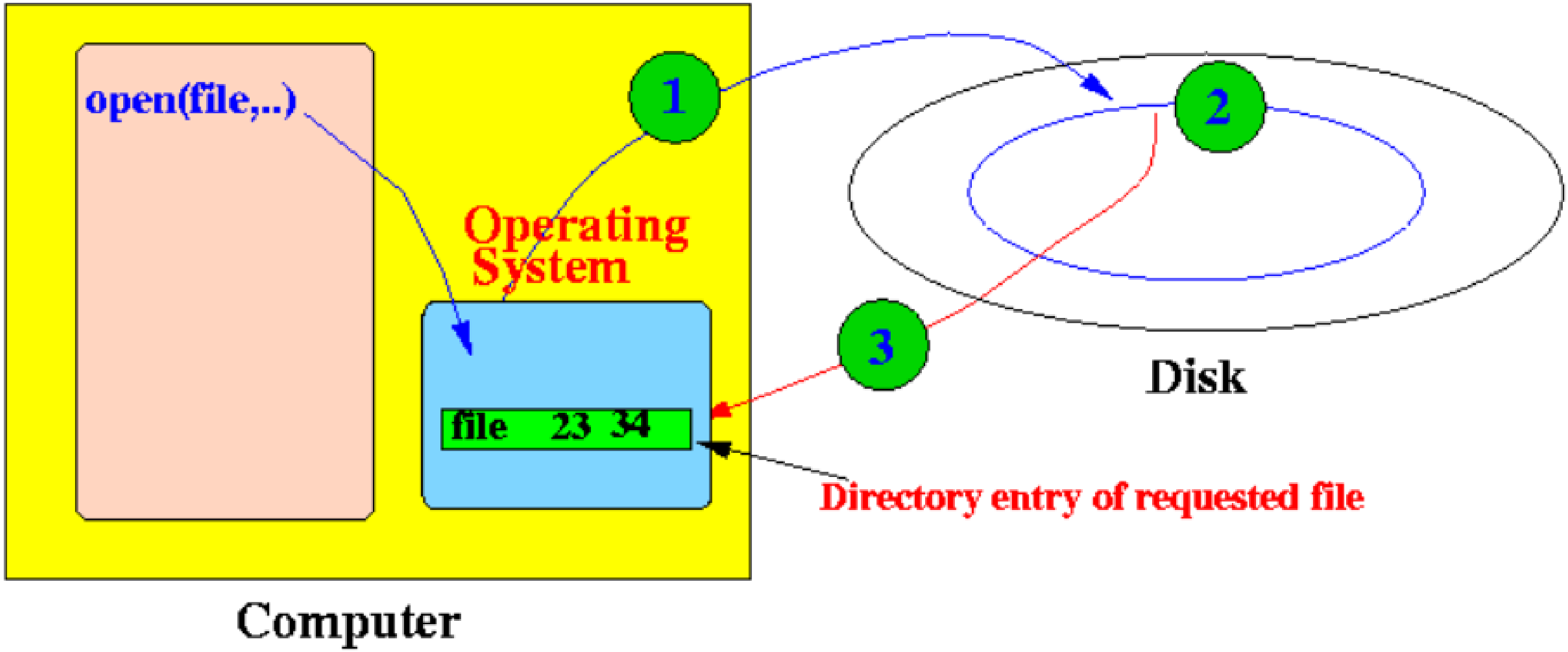

File System Workflow
APFS
Hard Drive
Partition
File System
Directory
Files

File Storage on Hard Disks

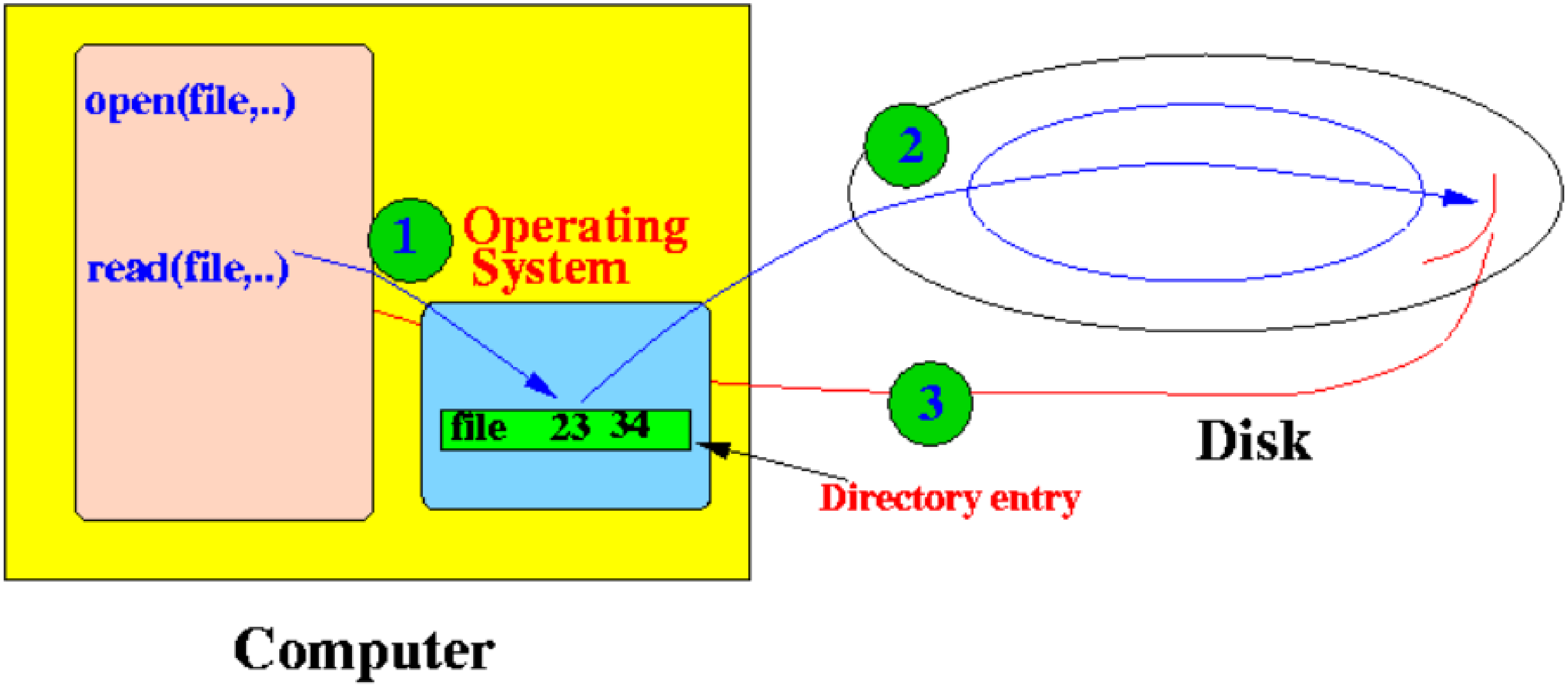
- Each computer system has a (built-in) file system
    - File systems used by Microsoft Windows are: FAT (File Allocation Table), FAT32 and NTFS
    - File systems used by Solaris UNIX are: UFS (UFS File System) and ZFS (ZFS File System)
- The file system always reserve a number of disk blocks (called: directory disk blocks) to store information about the files on the disk
    - The directory disk blocks contains a large number of directory entries

Accessing Data Stored in Files on a Hard Disk

1. Read directory disk blocks
2. Find directory entry for file
3. Read data blocks of file

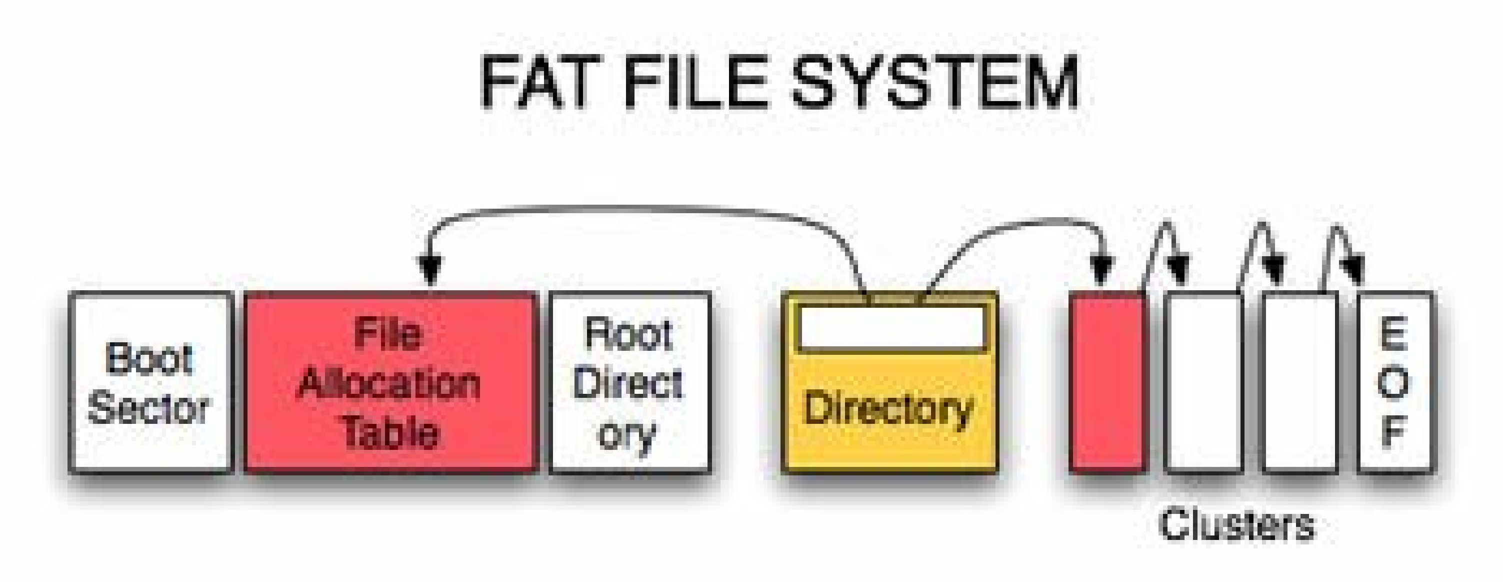

open(file,..)
1
2
Operating System
3
file 23 34
Disk
Directory entry of requested file
Computer

1. Find the block number to read
2. Locate the disk sectors of the block
3. Read data from disk at proper sector


open(file,..)
read(file,..)
1
Operating System
2
3
file 23 34
Directory entry
Disk
Computer

File Allocation Table (FAT)

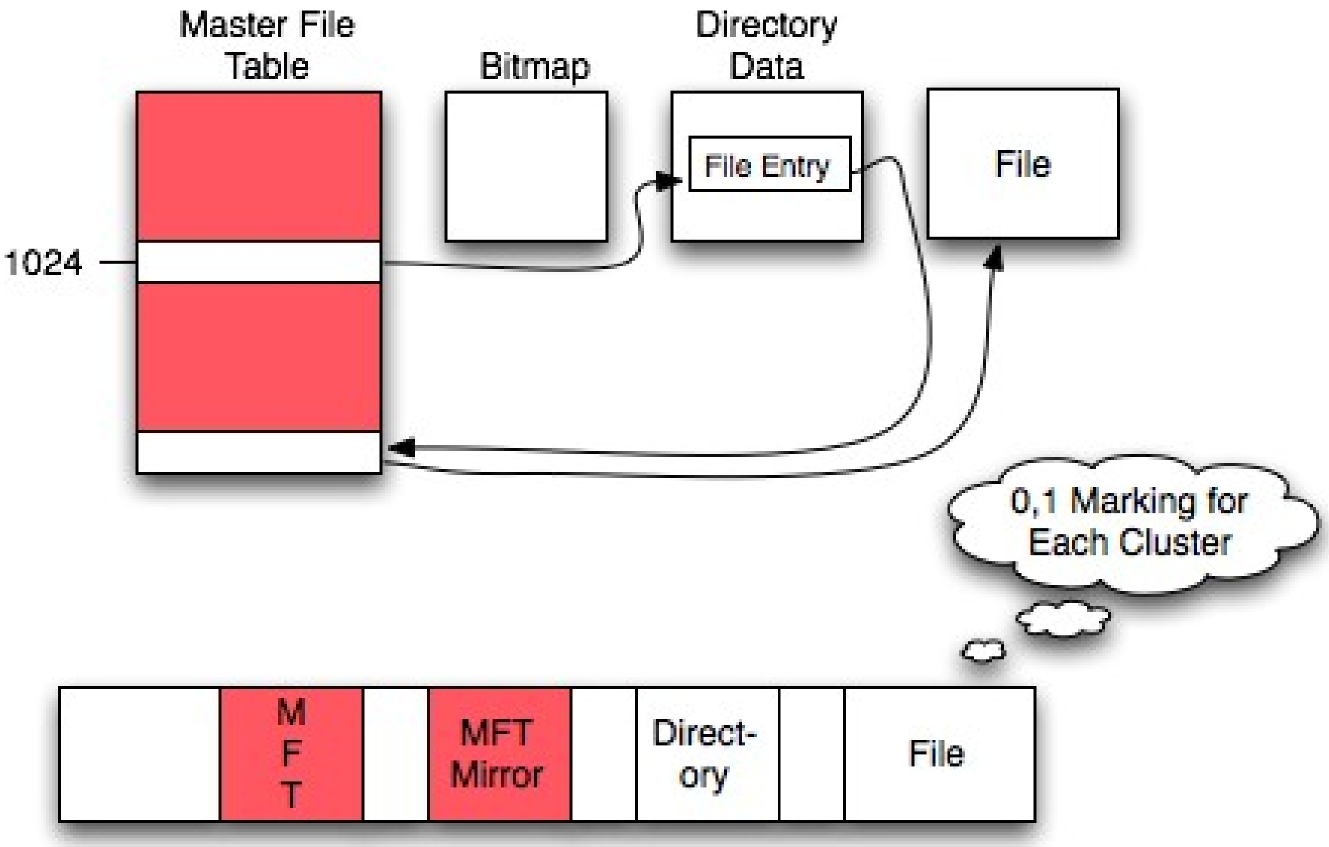
- Simple, handy and robust
- The file system originates from its prominent use of an index table that is allocated at the time of formatting
- Various versions of FAT file systems are FAT 12, FAT 16 and FAT 32


FAT FILE SYSTEM
Boot Sector
File Allocation Table
Root Direct ory
Directory
E O F
Clusters

NTFS

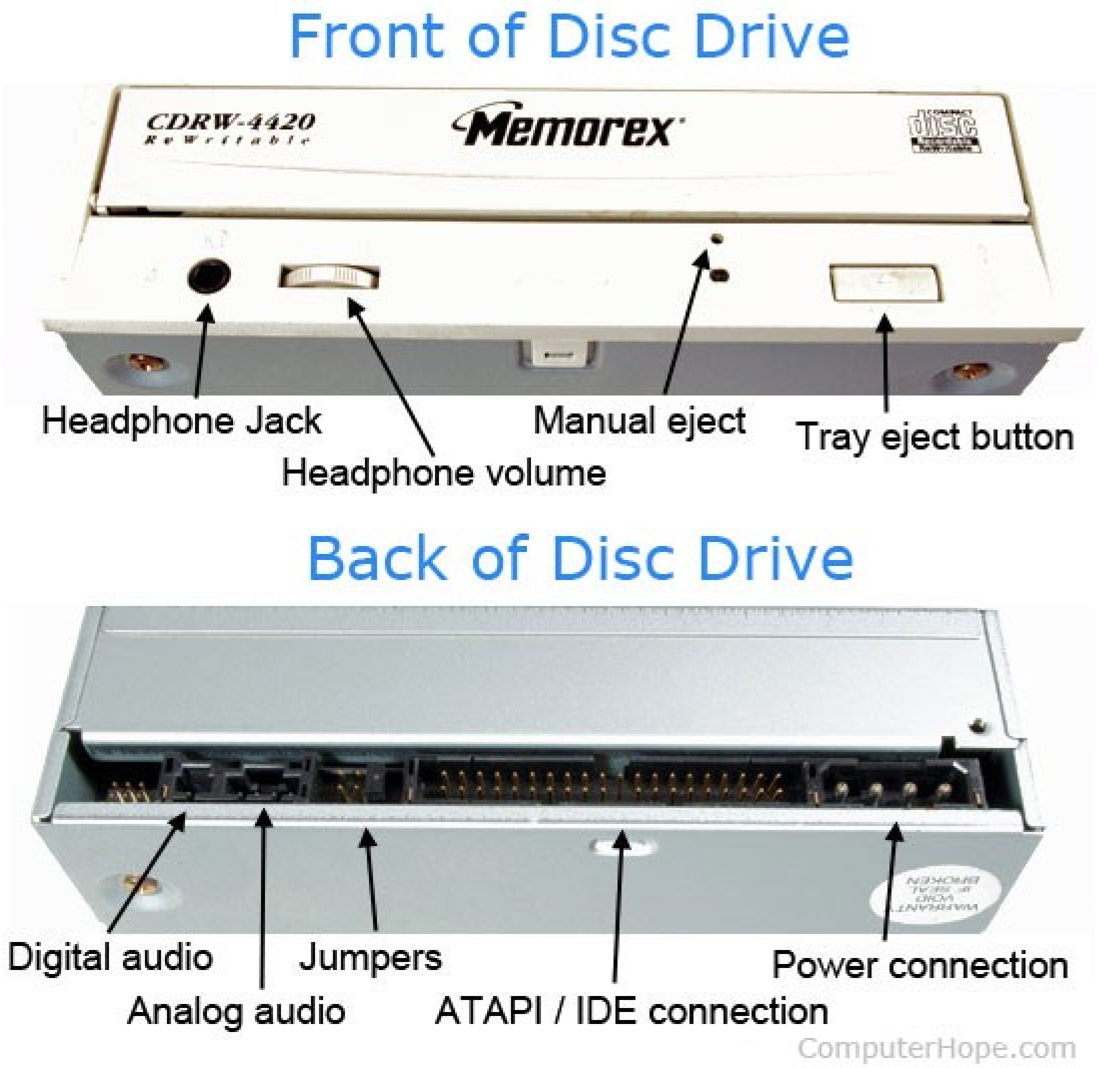
- The NTFS - **New Technology File System**
- While accessing (read) files on NTFS partitions on Mac-based systems, unable to transfer data from a Mac to that partition.
- NTFS is compatible with some media devices, but it is necessary to check the device's owner's manual to see if it's compatible.

NTFS FILE SYSTEM


Master File Table
Bitmap
Directory Data
File Entry
File
1024
0,1 Marking for Each Cluster
M F T
MFT Mirror
Direct-ory
File

Compact Disc Read Only Memory (CD-ROM)

Compact Disc Read Only Memory (CD-ROM)


Front of Disc Drive
CDRW-4420
ReWritable
Memorex
Headphone Jack
Manual eject
Tray eject button
Headphone volume
Back of Disc Drive
Digital audio
Jumpers
Power connection
Analog audio
ATAPI / IDE connection
ComputerHope.com

Compact Disc (CD) File System

- Features of **Compact Disc File System (CDFS)** are:
  - It is a file system for read-only and write-once CD-ROMs
  - It exports all tracks and boot images on a CD as normal files
  - CDFS provides a wide range of services which include creation, replacing, renaming, or deletion of files on write-once media
  - It uses a VCACHE driver to control the CD-ROM disc cache allowing for a smoother playback
  - It includes several disc properties like volume attributes, file attributes, and file placement
- sd

Types of CDFS

There are different versions of Compact Disk File System:

1. **Clustered operated system. (can be Global or Grid)**
2. **Flash operated**
3. **Object file system**
4. **Semantic**
5. **Steganographic process**
6. **Versioning**
7. **Synthetic operated system**

Advantages of CDFS

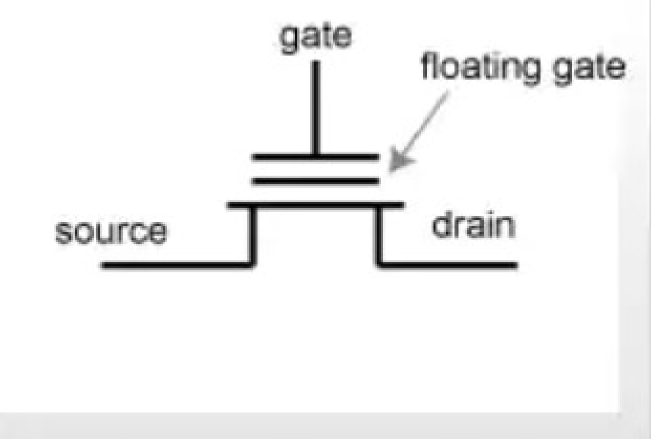
1. Very easy to transfer files from CDs to PC and vice versa.
2. CDFS supports cross-platform compatibility.
3. The CDs are very cheap to produce and so is the system.
4. The system uses a very fast means to transfer data.
5. Most computers read CDs so It is a huge success for Compact Disk File System.

Flash ROM

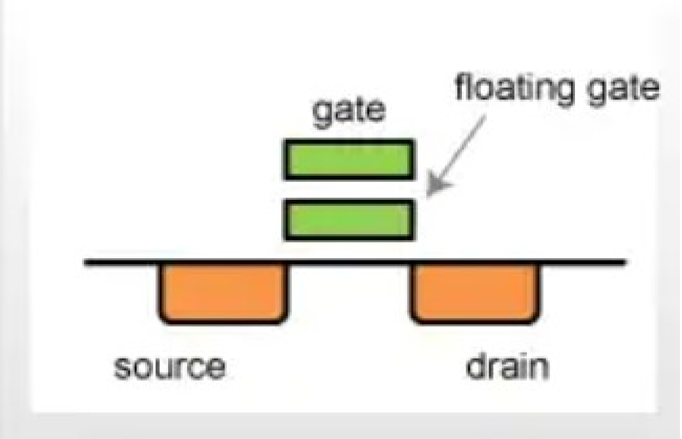
- A flash ROM is a type of non-volatile memories (flash)
- A flash ROM sets data of all cells (cell matrix) in a memory to 1 at once.
- That is, it gets electrons out from the floating gate of all the memory cells at a stretch, and leaves holes.
- ‘Flash’ is named from the image of momentary flash of lighting.

Memory Cell of a Flash ROM

- A floating gate is like a pocket for putting in coins.
- If an electron is put in there, the electron will stay in there.
- This electron remains into the floating gate even after a microcontroller is turned off. It is the basic mechanism of a non-volatile memory.


gate
floating gate
source
drain

The symbols of the cell of a flash ROM


floating gate
gate
source
drain

Thesectional view of the cell of a flash ROM

Flash Memory

Advantages

- Faster read and write compared to traditional hard disk drives.
- Smaller size.
- Less prone to damage.
- Cheaper than traditional drives in small storage capacities.
- Uses less power than traditional hard disk drives.

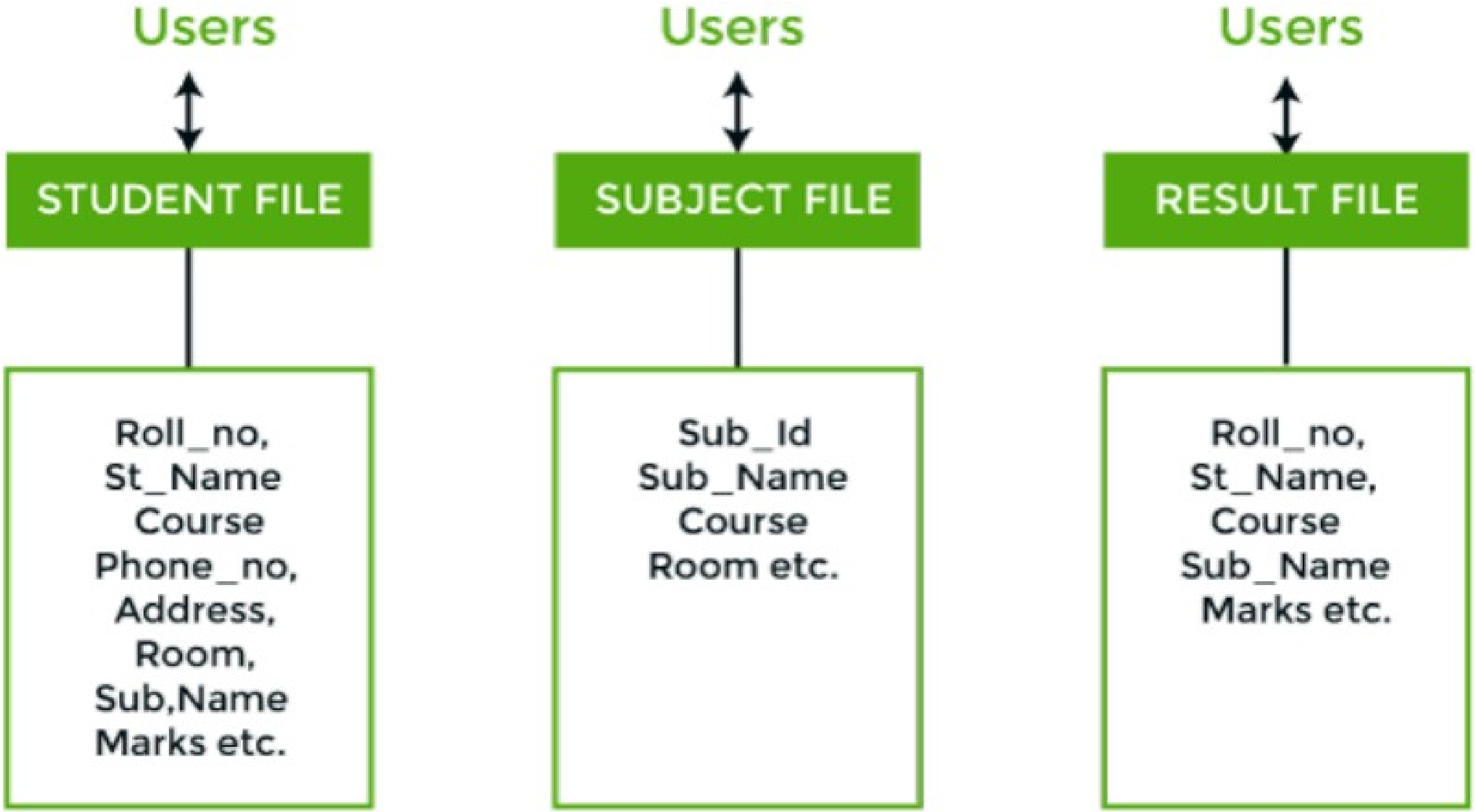
Disadvantages

- Flash memory cells have a limited number of write and erase cycles before failing.
- Smaller size devices, such as flash drives make them easier to lose
- May require a special version of a program to run on a flash-based drive to protect from prematurely wearing out the drive.

File System Vs Databases

- **File based systems** - traditional based approach, a decentralized approach was taken where each department stored and controlled its own data (data processing specialist)

- They create the necessary computer file structures, and manage the data within structures and design some application programs that create reports based on file data.



Users

STUDENT FILE

Roll_no,
St_Name
Course
Phone_no,
Address,
Room,
Sub,Name
Marks etc.

Users

SUBJECT FILE

Sub_Id
Sub_Name
Course
Room etc.

Users

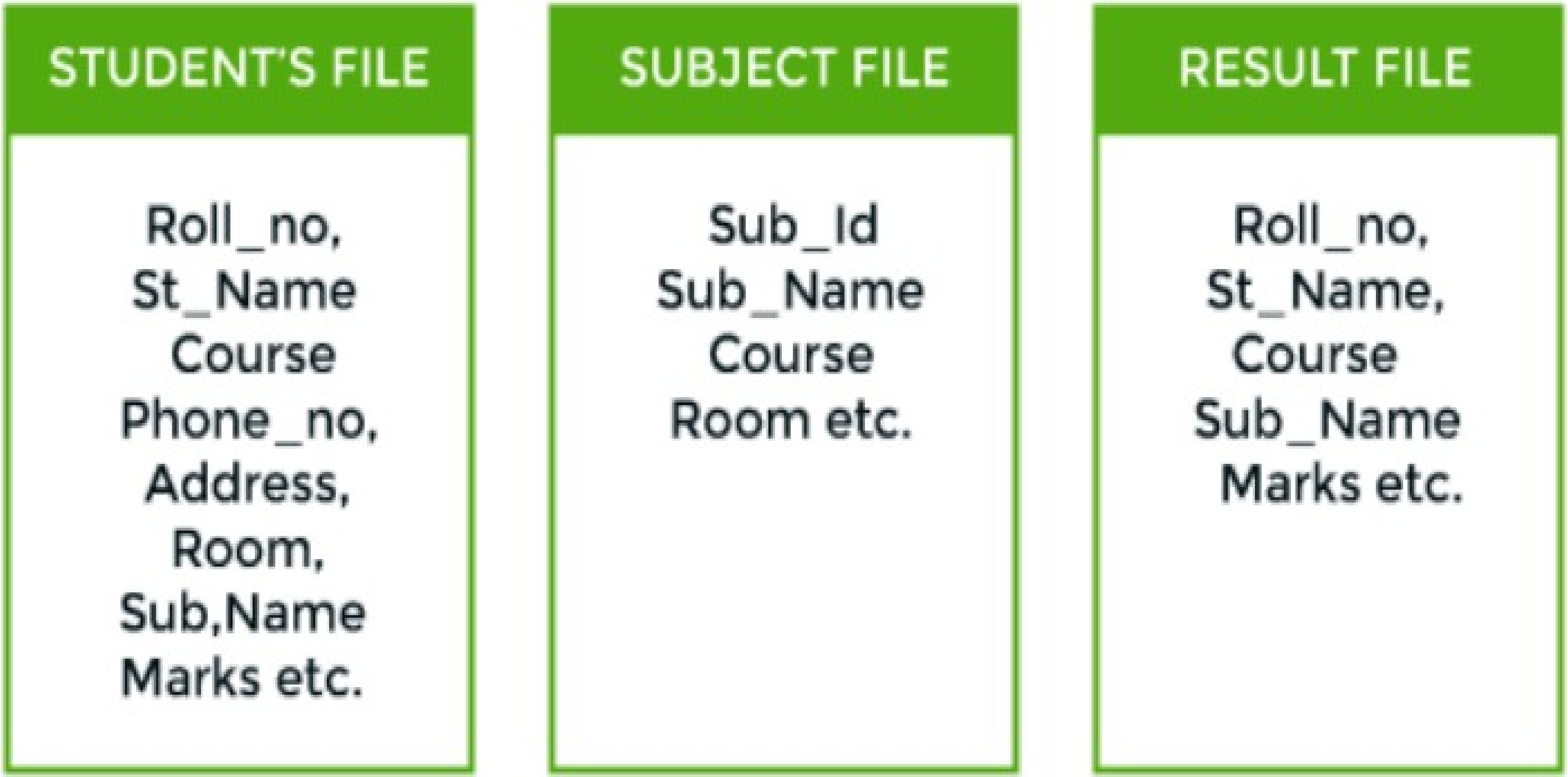
RESULT FILE

Roll_no,
St_Name,
Course
Sub_Name
Marks etc.

DataBase Management System (DBMS)

- **A database approach** - a well-organized collection of data that are related in a meaningful way which can be accessed by different users but stored only once in a system.

- The DBMS system operations are:
    - **Insertion,**
    - **Deletion,**
    - **Selection,**
    - **Sorting etc.**



STUDENT'S FILE

Roll_no,
St_Name
Course
Phone_no,
Address,
Room,
Sub,Name
Marks etc.

SUBJECT FILE

Sub_Id
Sub_Name
Course
Room etc.

RESULT FILE

Roll_no,
St_Name,
Course
Sub_Name
Marks etc.

File System and DBMS

	File System	DBMS
Meaning	The user has to write the procedures for managing the database	The user is not required to write the procedures
Sharing of data	It may be of different formats, so it isn't easy to share data.	Data sharing is easy
Data	Provides the detail of the data representation and storage of data	An abstract view of data that hides the details
Security and Protection	It isn't easy to protect a file under the file system	Good protection mechanism
Recovery Mechanism	The file system doesn't have a crash mechanism	DBMS protects the user from system failure

	File System	DBMS
Manipulation Techniques	Can't efficiently store and retrieve the data	Sophisticated techniques to store and retrieve the data
Concurrency Problems	Concurrent access has many problems like redirecting the file while deleting some information or updating some information	Concurrent access of data using some form of locking
Cost	Cheaper to design	Expensive to design
Data Redundancy and Inconsistency	There exists a lot of duplication of data	Problems are controlled
Structure	Simple structure	Complex to design

	File System	DBMS
Data Redundancy and Inconsistency	There exists no Data Independence	Data Independence exists, and it can be of two types. Logical Data Independence Physical Data Independence
Integrity Constraints	Difficult to implement	Easy to apply
Data Models	No concept of data models exists	Three types of data models exist: Hierarchal, Network and Relational data models
Flexibility	Less	More
Examples	Cobol, C++ etc	Oracle, SQL Server, Sybase etc

What is a file system?

- Speaking broadly, a file system is the logical means for an operating system to store and retrieve data on the computers hard disks, be they local drives, network-available volumes, or exported shares in a storage area network (SAN)

What is a file system?

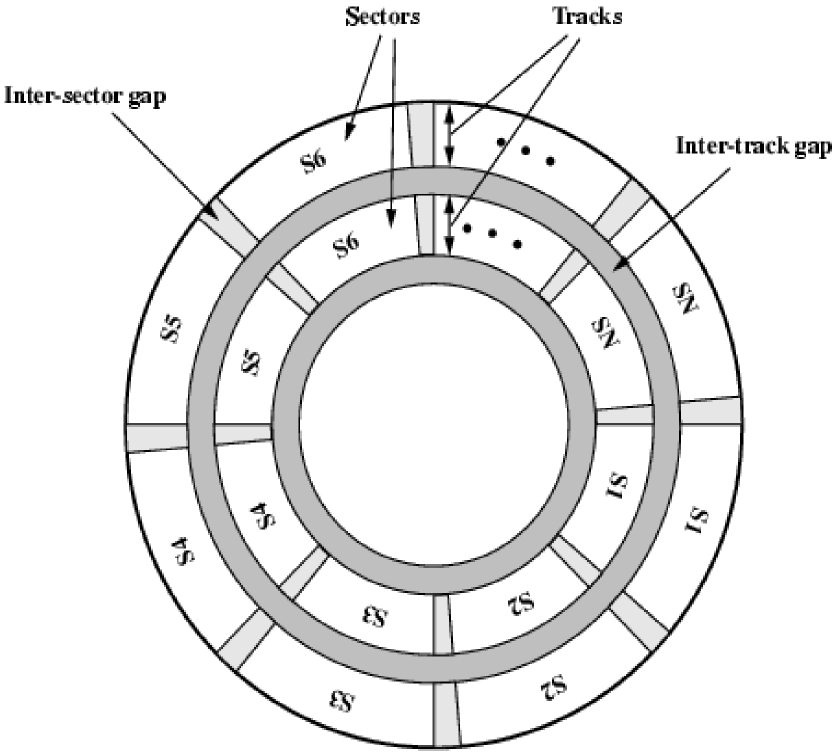
A file system can refer to the methods and data structures that an operating system uses to keep track of files on a disk or partition

- Linux keeps regular files and directories on block devices such as disks
- A Linux installation may have several physical disk units, each containing one or more file system types
- Partitioning a disk into several file system instances makes it easier for administrators to manage the data stored there

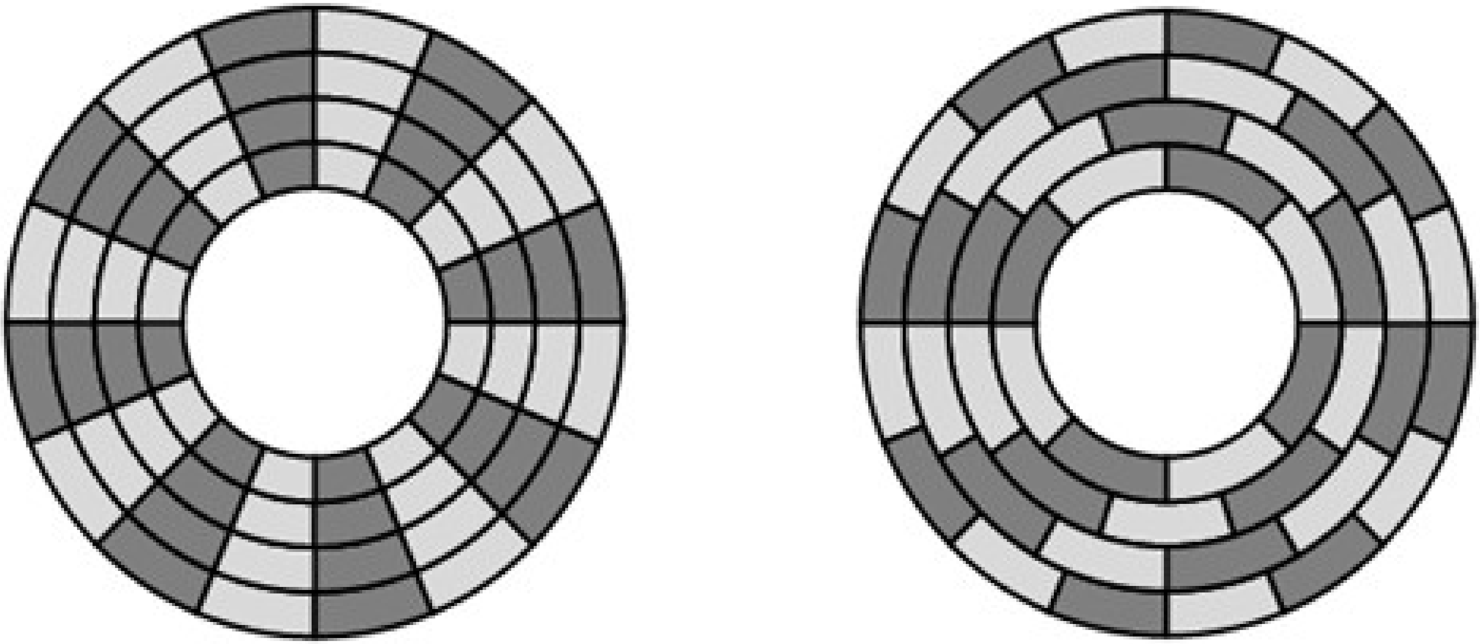
Magnetic Disk

Data Organization on Disk

- Concentric rings called tracks
  - Gaps between tracks
  - Same number of bits per track
  - Constant angular velocity
- Tracks divided into sectors
- Minimum block size is one sector
- Disk rotate at const. angular velocity
  - Gives pie shaped sectors
- Individual tracks and sectors addressable


Sectors
Tracks
Inter-sector gap
Inter-track gap
S6
S6
SN
SN
S5
S5
S1
S1
S4
S4
S3
S3
S2
S2

Multi Zone Recording Disks



Disk storage capacity in a CAV system is limited by the maximum recording density that can be achieved on the innermost track. (constant angular velocity)

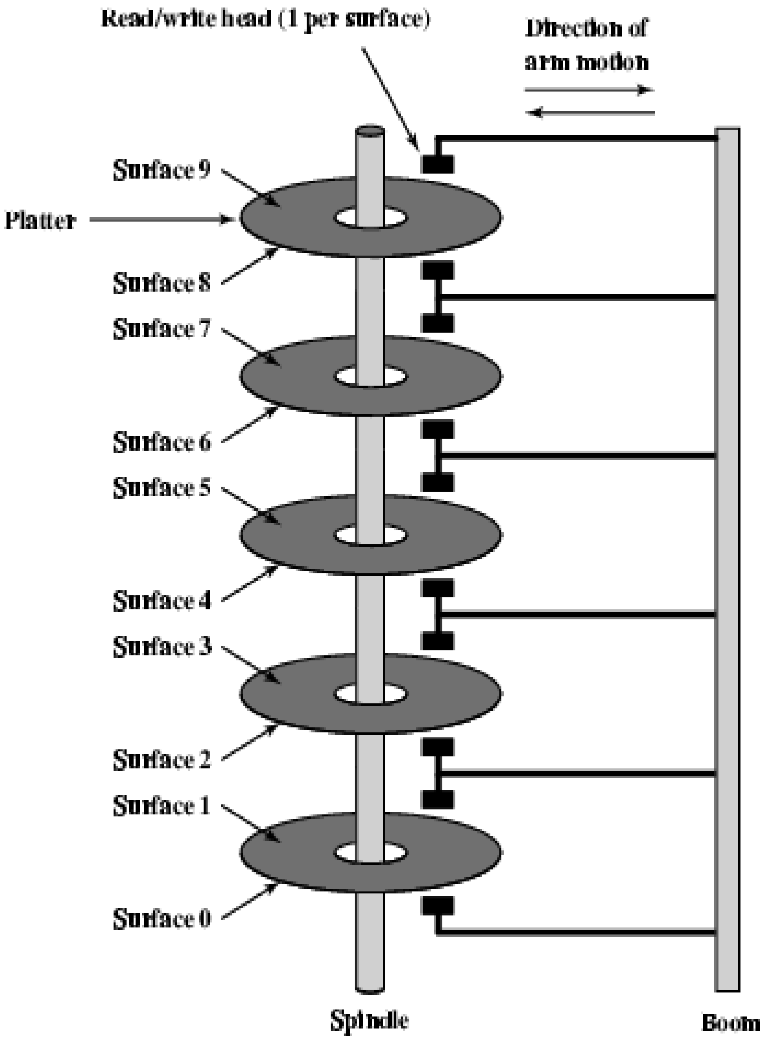
Read and Write Mechanisms-1

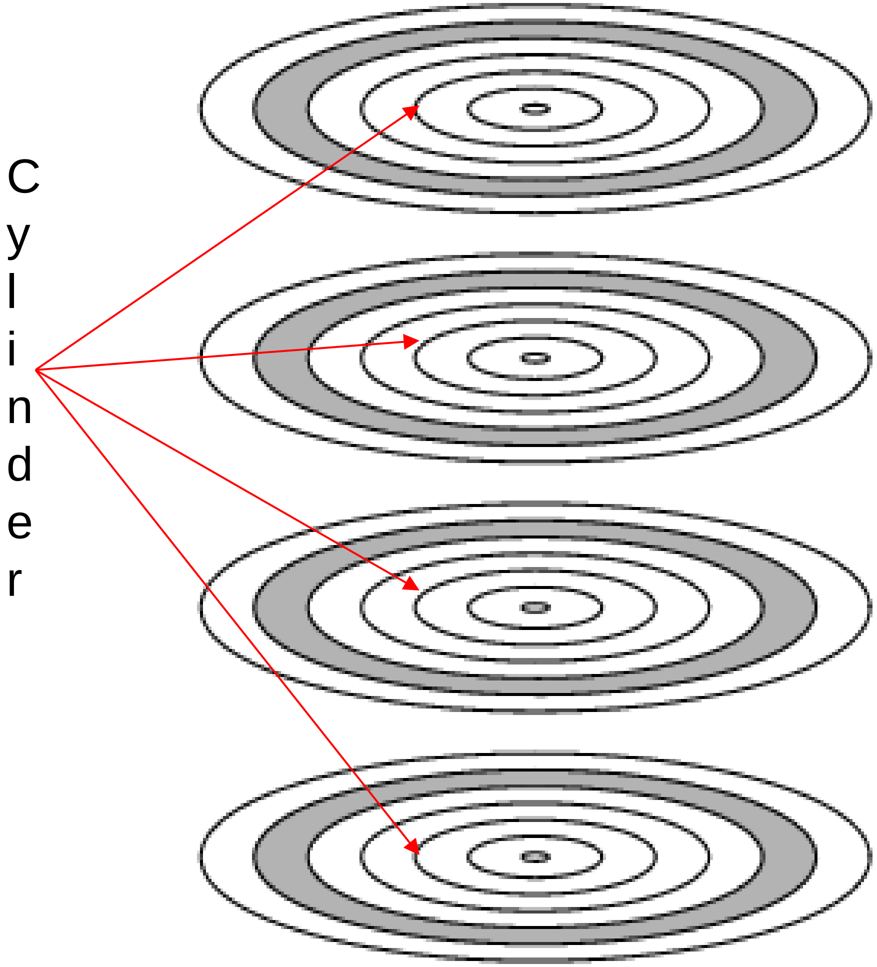
- Recording & retrieval of data via conductive coil which is called a head
- May be single read/write head or separate ones
- During read/write, head is stationary, platter rotates
- Write
  - Electricity flowing through coil produces magnetic field
  - Electric pulses sent to head
  - Magnetic pattern recorded on surface below

Disk Characteristics

- Fixed (rare) or movable head
  - Fixed head
    - One read/write head per track mounted on fixed ridged arm
  - Movable head
    - One read/write head per side mounted on a movable arm
- Removable or fixed disk
- Single or double (usually) sided
- Head mechanism
  - Contact (Floppy), Fixed gap, Flying (Winchester)
- Single or multiple platter

Multiple Platters Tracks and Cylinders


Read/write head (1 per surface)
Direction of arm motion
Surface 9
Platter
Surface 8
Surface 7
Surface 6
Surface 5
Surface 4
Surface 3
Surface 2
Surface 1
Surface 0
Spindle
Boom



Capacity

- Vendors express capacity in units of gigabytes (GB), where 1 GB =10^9 Byte
- Capacity is determined by these technology factors:
  - Recording density (bits/in): number of bits that can be squeezed into a 1 inch segment of a track.
  - Track density (tracks/in): number of tracks that can be squeezed into a 1 inch radial segment.
  - Areal density (bits/in2): product of recording and track density.
- Modern disks partition tracks into disjoint subsets called recording zones

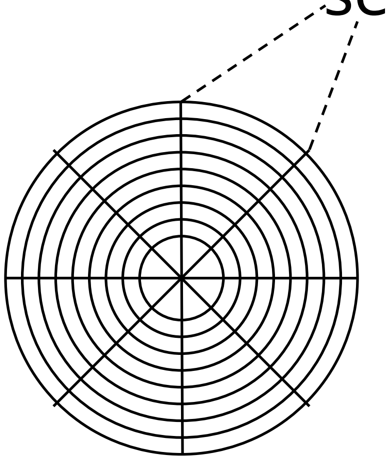
Computing Disk Capacity

- Capacity =(# bytes/sector) x (avg. # sectors/track) x

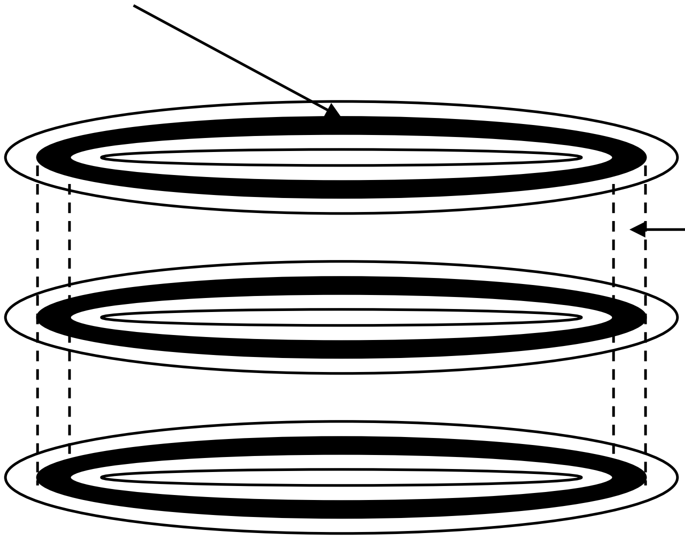
  (# tracks/surface) x (# surfaces/platter) x

  (# platters/disk)
- Example:
  - 512 bytes/sector, 300 sectors/track (average)
  - 20,000 tracks/surface, 2 surfaces/platter
  - 5 platters/disk
  - Capacity = 512 x 300 x 20000 x 2 x 5 = 30.72GB

What is a file system?


Overhead view
sector

Disk blocks are composed of one or more contiguous sectors


track
cylinder

The same track on each platter in a disk makes a cylinder; partitions are groups of contiguous cylinders

What is a file system?

- File system instances reside on partitions
- Partitioning is a means to divide a single hard drive into many logical drives
- A partition is a contiguous set of blocks on a drive that are treated as an independent disk
- A partition table is an index that relates sections of the hard drive to partitions

The UNIX File System

- The root directory has many subdirectories. The following table describes some of the subdirectories contained under root

Directory	Content
/bin	Common programs, shared by the system, the system administrator and the users.
/dev	Contains references to all the CPU peripheral hardware, which are represented as files with special properties.
/etc	Most important system configuration files are in /etc, this directory contains data similar to those in the Control Panel in Windows
/home	Home directories of the common users.
/lib	Library files, includes files for all kinds of programs needed by the system and the users.
/sbin	Programs for use by the system and the system administrator.
/tmp	Temporary space for use by the system, cleaned upon reboot, so don't use this for saving any work!
/usr	Programs, libraries, documentation etc. for all user-related programs.
/var	Storage for all variable files and temporary files created by users, such as log files, the mail queue, the print spooler area, space for temporary storage of files.

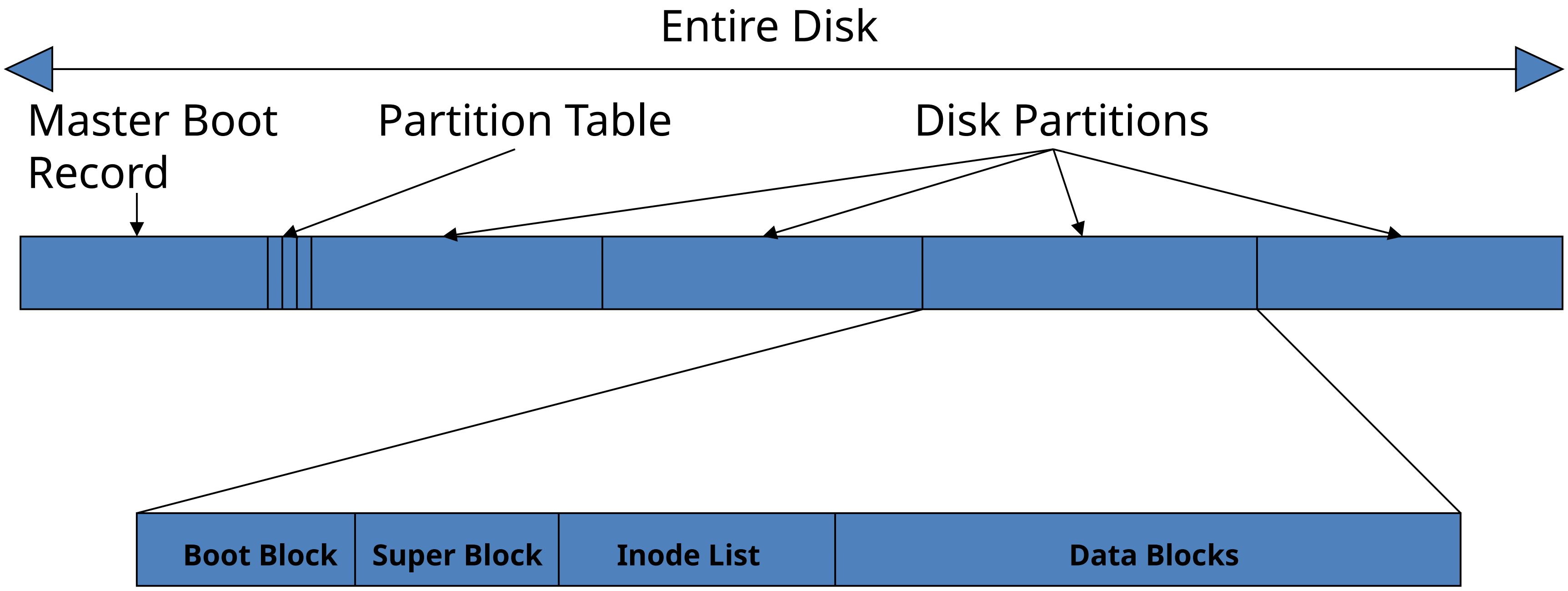
UNIX and POSIX file attributes

- **File type** : Type of file
- **Access permission** : The file access permission (owner, group, others)
- **Hard link count** : Number of hard links of a file
- **UID** : The file owner user ID
- **GID** : The file group ID
- **File size** : The file size in bytes
- **Last access time** : The time the file was last accessed
- **Last modify time** : The time the file was last modified
- **Last change time** : The time the file access permission UID ,GID or hard link count was last changed
- **Inode number** : The system inode number of the file

INTERNAL REPRESENTATION OF FILES

- Inodes
- Structure of a regular file
- Directories
- Conversion of a path name to an inode
- Super block
- Inode assignment to a new file
- Allocation of disk blocks
- Other file types

File System Implementation
(Disk Layout)


Entire Disk
Master Boot Record
Partition Table
Disk Partitions
Boot Block
Super Block
Inode List
Data Blocks

A Possible File System Instance Layout

File system layout

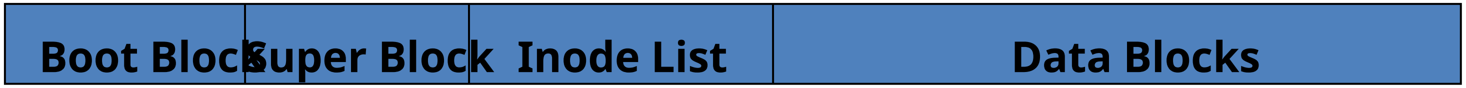
- The central structural concepts of a file system type are:

  - Boot Block

  - Super Block

  - Inode List

  - Data Block


Boot Block
Super Block
Inode List
Data Blocks

Boot Block

Boot Block:

- Occupies the beginning of a file system

- Typically residing at the first sector, it may also contain the bootstrap code that is read into the machine at boot time

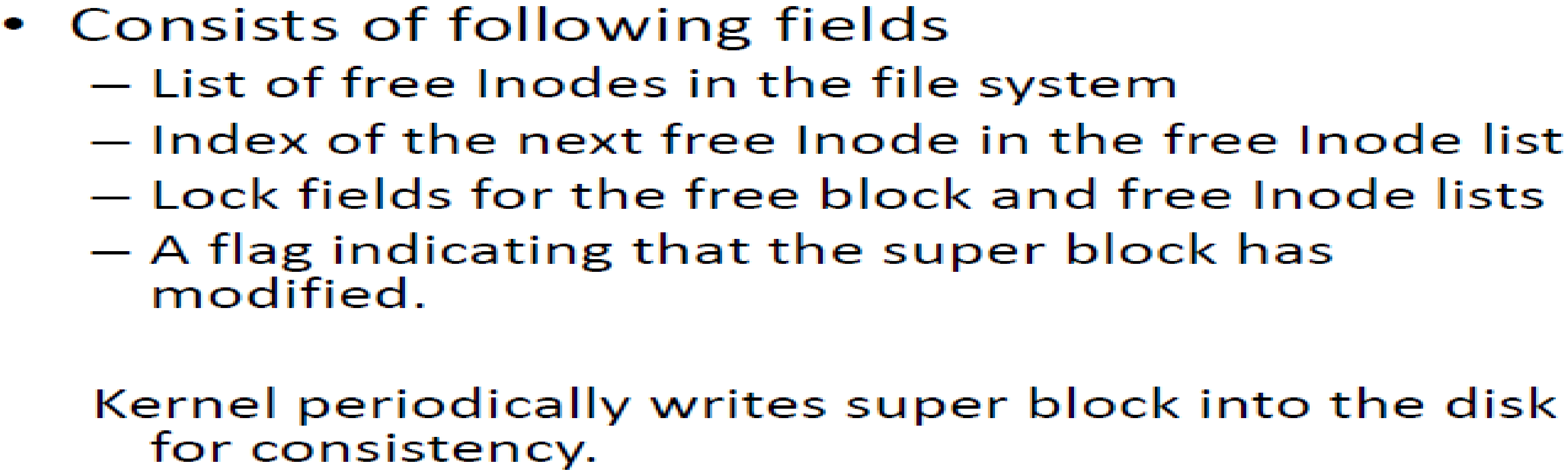
- Although only one boot block is required to boot the system, every file system may contain a boot block

Boot Block

Super Block

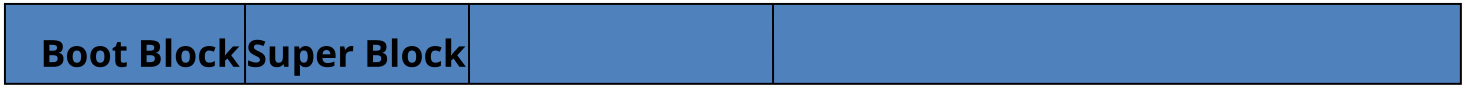
Super Block:

- Describes the state of a file system
- How large it is
- How many files it can store
- Where to find free space in the file system



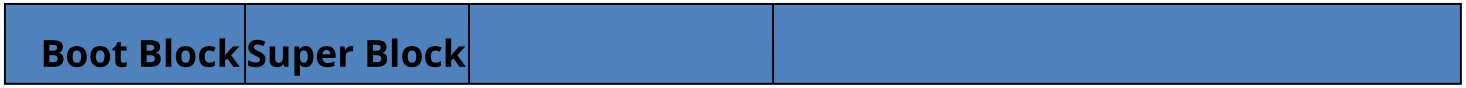
- Consists of following fields
  - List of free Inodes in the file system
  - Index of the next free Inode in the free Inode list
  - Lock fields for the free block and free Inode lists
  - A flag indicating that the super block has modified.

Kernel periodically writes super block into the disk for consistency.



Super Block

- Consists of following fields
    - Size of the file system
    - Number of free blocks in the file system
    - List of free blocks available in the file system
    - Index of the next free block in the free block list
    - Size of the Inode list
    - Number of free Inodes in the file system



Boot Block **Super Block**

Inode List

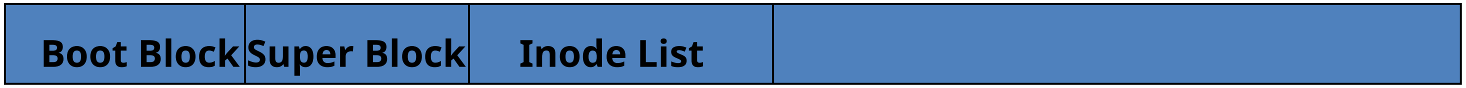
Inode List:

- An inode is the internal representation of a file

contains the description of the disk layout of the file data

- file owner
- permissions

- The inode list contains all of the inodes present in an instance of a file system


Boot Block
Super Block
Inode List

Data Blocks

Data Blocks:

- Contain the file data in the file system
- Additional administrative data
- An allocated data block can belong to one and only
  one file in the file system

Boot Block	**Super Block**	**Inode List**	**Data Blocks**

What is a file system?

So what is a file system?

- A file system is a set of abstract data types that are implemented for the storage, hierarchical organization, manipulation, navigation, access, and retrieval of data

Inodes in UNIX system

- UNIX system has an inode table which keeps track of all files
- Each entry in inode table is an inode record
- Inode record contains all attributes of file including inode number and physical address of file
- Information of a file is accessed using its inode number
- Inode number is unique within a file system
- A file record is identified by a file system ID and inode number
- Inode record doesnot contain the name of the file
- The mapping of filenames to inode number is done via directory files

What is a file system?

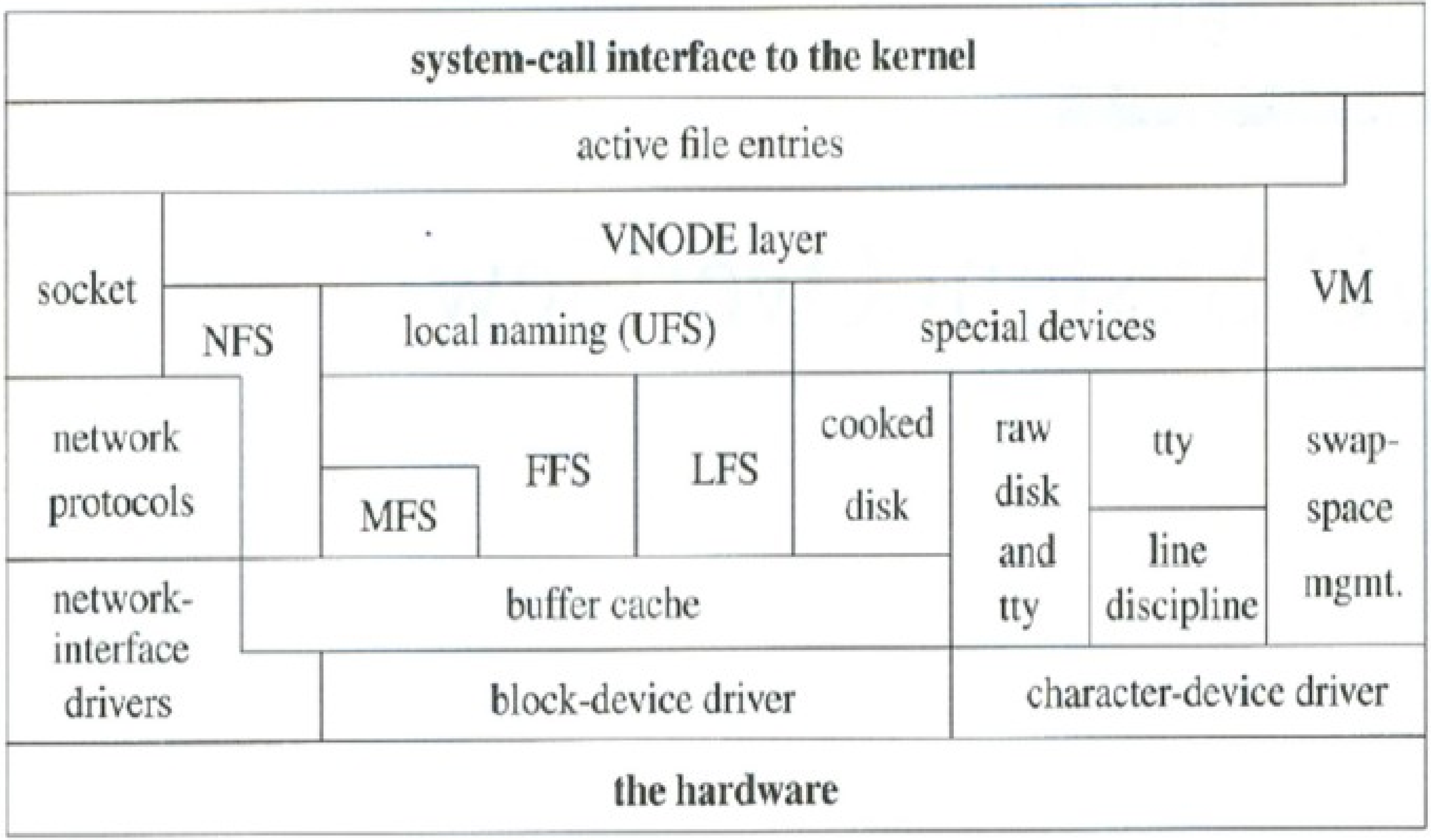
- The kernel deals on a logical level with file systems rather than with disks
- The separate file systems that the system may use are not accessed by device identifiers
- Instead they are combined into a single hierarchical tree structure that represents the file systems as one whole single entity

File system consistency check

- During bootstrap operation, all the file systems are checked for self – consistency and attached to the root system
  - UNIX command: fsck

- To find the available space on the mounted file systems use UNIX command df (disc free space)

Kernel I/O structure



system-call interface to the kernel
active file entries
socket
VNODE layer
VM
NFS
local naming (UFS)
special devices
network protocols
FFS
LFS
MFS
cooked disk
raw disk and tty
tty
line discipline
swap-space mgmt.
network-interface drivers
buffer cache
block-device driver
character-device driver
the hardware

UFS

- Fast File System (FFS): It assumes that the average file size is small and that some files (those in the same directory) will be accessed all at once. Given these assumptions, it divides the disk surface in different cylinder groups and lays out blocks that may be accessed together in the same group. This reduces expensive seeks.
- Log-structured File System (LFS): It assumes that the system bottleneck are writes to the disk, so it packs all writes in a concrete area of the disk, treating the overall space as an append-only log (no seeks required).
- Memory File System (MFS): Reserves a contiguous region of memory and lays out blocks in it. This is why MFS is considered inefficient, because it is using the concepts of an on-disk file-system over memory.

I/O in Unix - Devices

- I/O system used for
  - I/O devices, Network communication and Virtual Memory (Swap space)

```
[SysPgm@labserver ~] $  ls  -l   /dev/ ↵
```

  - Observe File types
    - c for Character device driver
    - b for Block device driver
    - l for symbolic link
  - Observe file size [Replaced by 2 integer numbers]
    - Major device number and minor device number

I/O in Unix - Devices

- Block Devices and Character Devices
  - Character devices – i.e. devices treated as byte streams
    - Include terminals, line printers, main memory apart from disks.
    - Unstructured or raw interface – no buffering by kernel.
  - Block Devices – i.e. block-addressable devices
    - E.g. Disks and Tapes
    - Structured interface – block-sized accesses – buffering done by kernel.

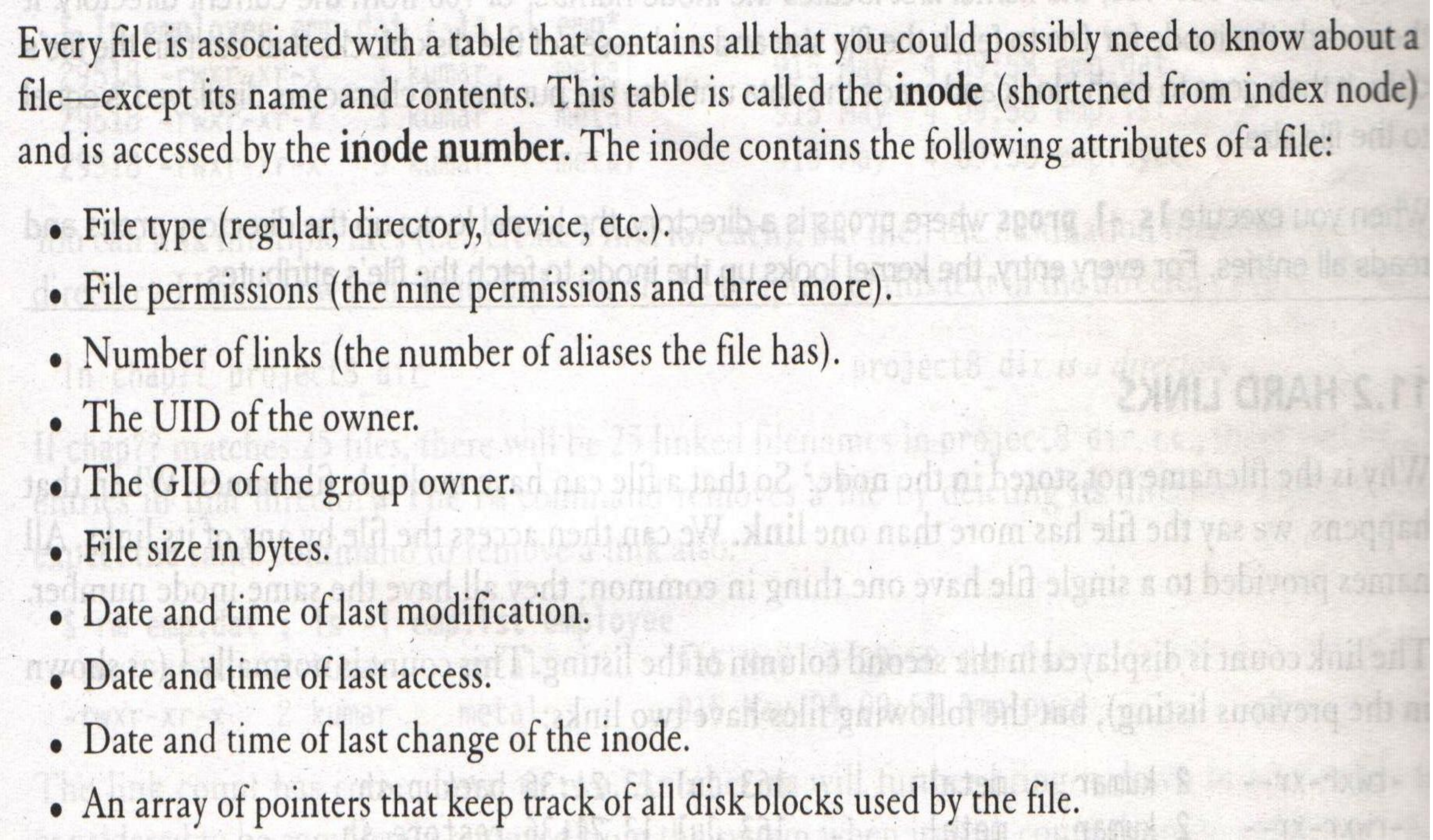
I/O in Unix - Devices

Device Identification – major device #, minor device #

- Major Device # refers to the type of device and hence is used to identify a device driver.
  - When a device file is opened, Linux examines its major number and forwards the call to the driver registered for that device.
- Minor Device # refers to a specific device and is interpreted by a device driver.
  - Minor number is used to identify the specific device instance if the driver controls more than one device of a type.

– Block Device table and Character Device Table

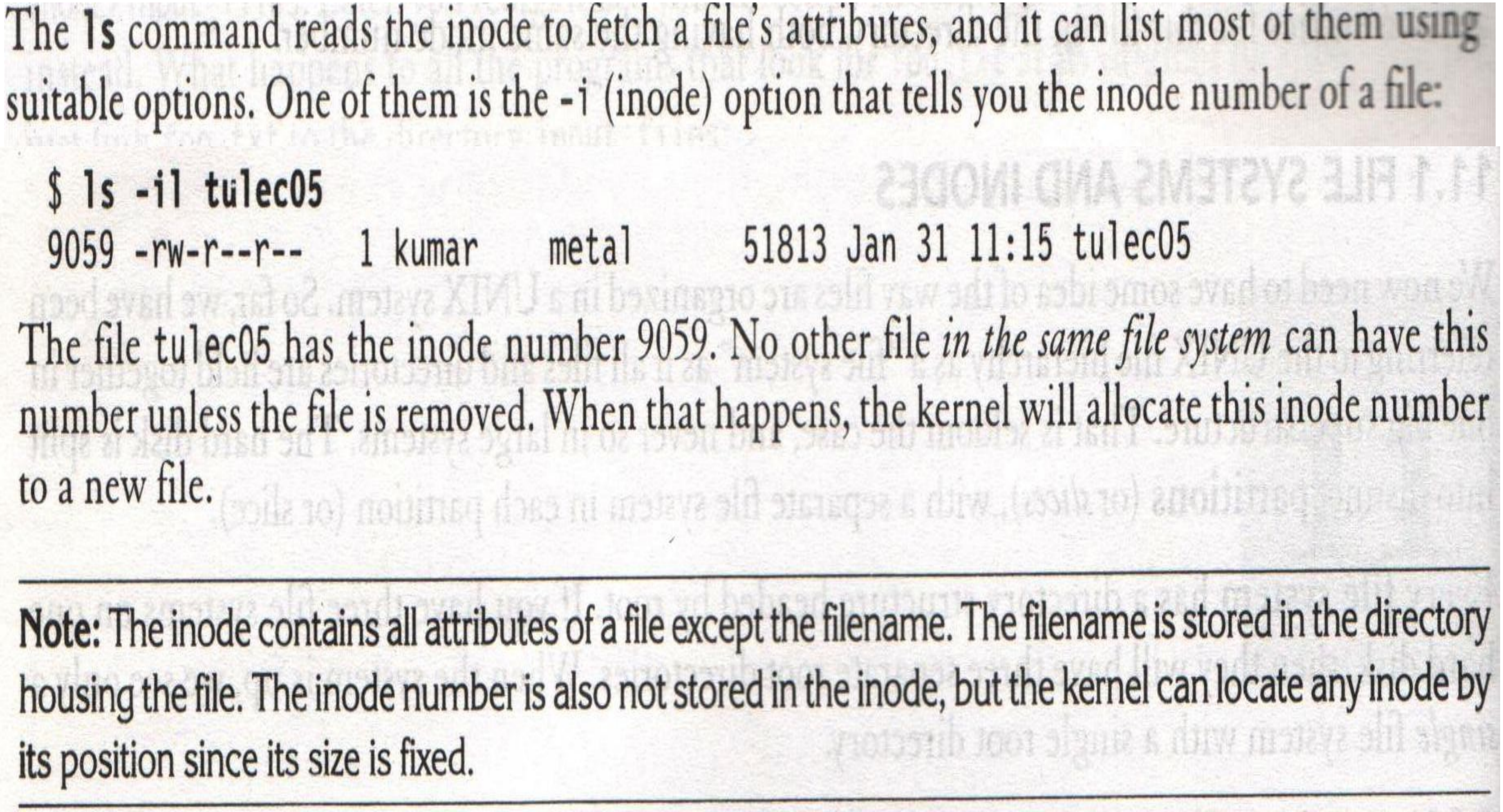
File Systems and inodes



Every file is associated with a table that contains all that you could possibly need to know about a file—except its name and contents. This table is called the **inode** (shortened from index node) and is accessed by the **inode number.** The inode contains the following attributes of a file:

- File type (regular, directory, device, etc.).
- File permissions (the nine permissions and three more).
- Number of links (the number of aliases the file has).
- The UID of the owner.
- The GID of the group owner.
- File size in bytes.
- Date and time of last modification.
- Date and time of last access.
- Date and time of last change of the inode.
- An array of pointers that keep track of all disk blocks used by the file.

File Systems and inodes



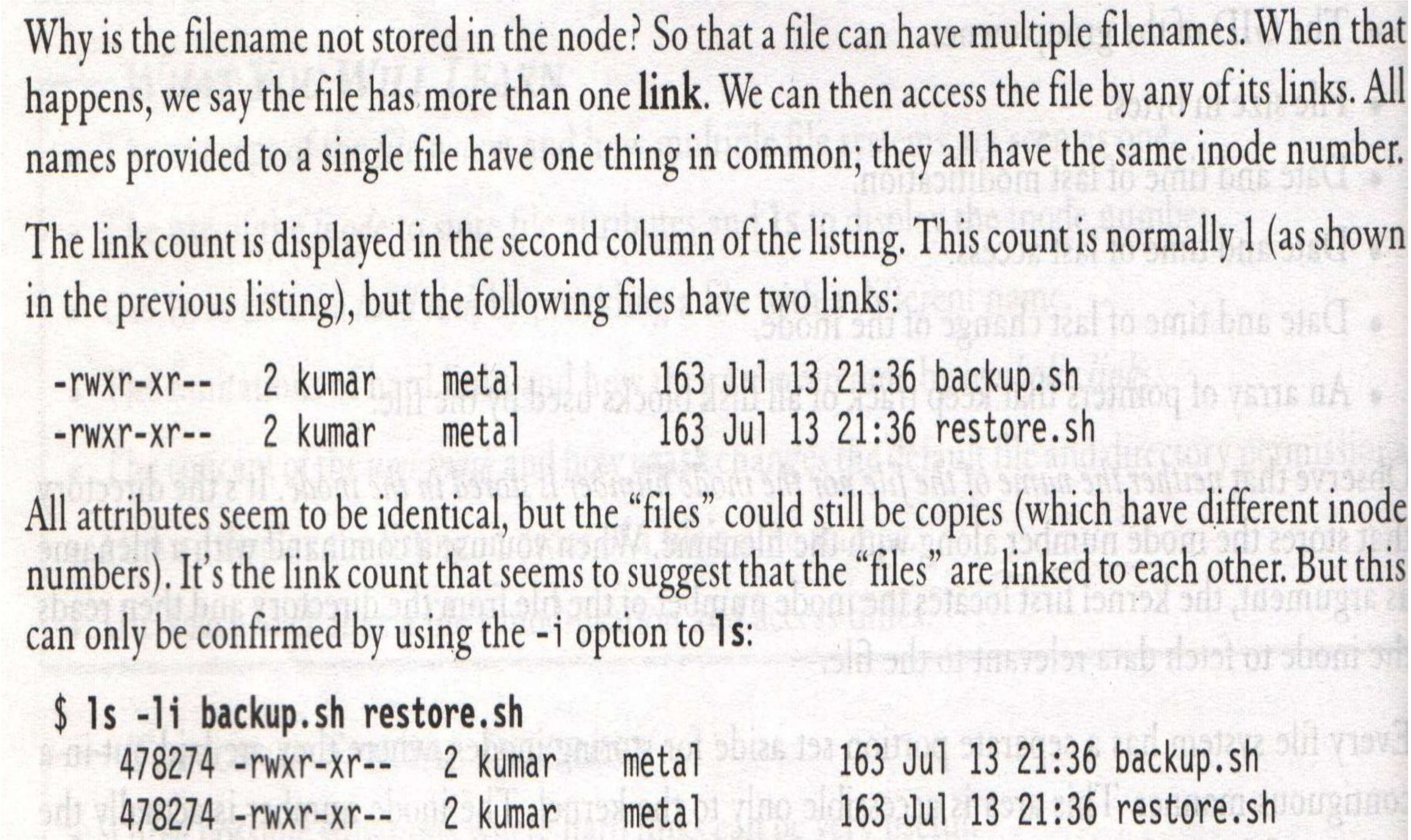
The `ls` command reads the inode to fetch a file's attributes, and it can list most of them using suitable options. One of them is the `-i` (inode) option that tells you the inode number of a file:

```
$ ls -il tulec05
9059 -rw-r--r--   1 kumar    metal       51813 Jan 31 11:15 tulec05
```

The file `tulec05` has the inode number 9059. No other file *in the same file system* can have this number unless the file is removed. When that happens, the kernel will allocate this inode number to a new file.

Note: The inode contains all attributes of a file except the filename. The filename is stored in the directory housing the file. The inode number is also not stored in the inode, but the kernel can locate any inode by its position since its size is fixed.

Hard Links



Why is the filename not stored in the node? So that a file can have multiple filenames. When that happens, we say the file has more than one **link**. We can then access the file by any of its links. All names provided to a single file have one thing in common; they all have the same inode number.

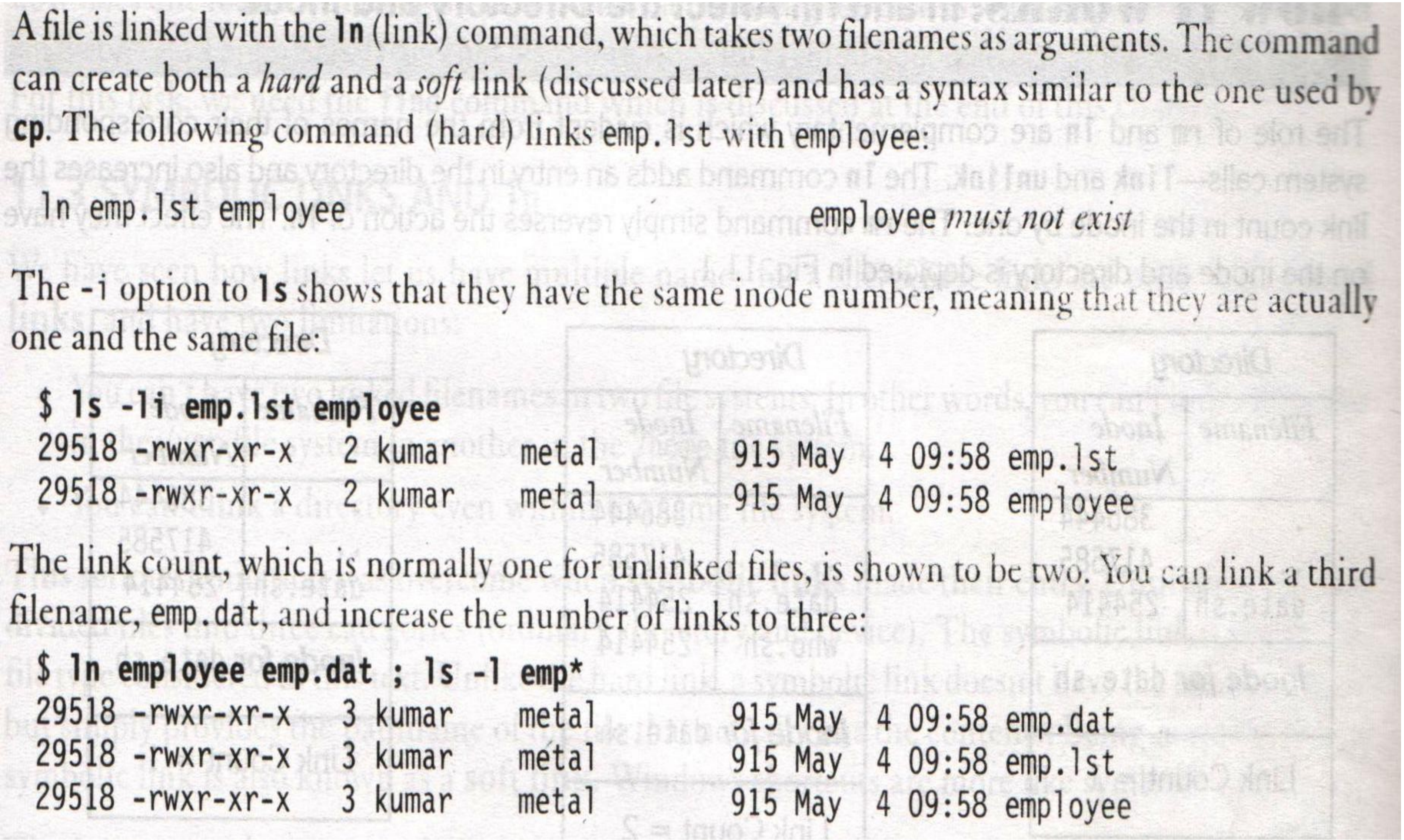
The link count is displayed in the second column of the listing. This count is normally 1 (as shown in the previous listing), but the following files have two links:

```
-rwxr-xr--   2 kumar   metal          163 Jul 13 21:36 backup.sh
-rwxr-xr--   2 kumar   metal          163 Jul 13 21:36 restore.sh
```

All attributes seem to be identical, but the "files" could still be copies (which have different inode numbers). It's the link count that seems to suggest that the "files" are linked to each other. But this can only be confirmed by using the `-i` option to `ls`:

```
$ ls -li backup.sh restore.sh
    478274 -rwxr-xr--   2 kumar   metal          163 Jul 13 21:36 backup.sh
    478274 -rwxr-xr--   2 kumar   metal          163 Jul 13 21:36 restore.sh
```

ln: Creating Hard Links



A file is linked with the **ln** (link) command, which takes two filenames as arguments. The command can create both a *hard* and a *soft* link (discussed later) and has a syntax similar to the one used by **cp**. The following command (hard) links `emp.lst` with `employee`:

```
ln emp.lst employee                                    employee must not exist
```

The `-i` option to **ls** shows that they have the same inode number, meaning that they are actually one and the same file:

```
$ ls -li emp.lst employee
29518 -rwxr-xr-x   2 kumar    metal          915 May  4 09:58 emp.lst
29518 -rwxr-xr-x   2 kumar    metal          915 May  4 09:58 employee
```

The link count, which is normally one for unlinked files, is shown to be two. You can link a third filename, `emp.dat`, and increase the number of links to three:

```
$ ln employee emp.dat ; ls -l emp*
29518 -rwxr-xr-x   3 kumar    metal          915 May  4 09:58 emp.dat
29518 -rwxr-xr-x   3 kumar    metal          915 May  4 09:58 emp.lst
29518 -rwxr-xr-x   3 kumar    metal          915 May  4 09:58 employee
```

WHERE TO USE HARD LINKS

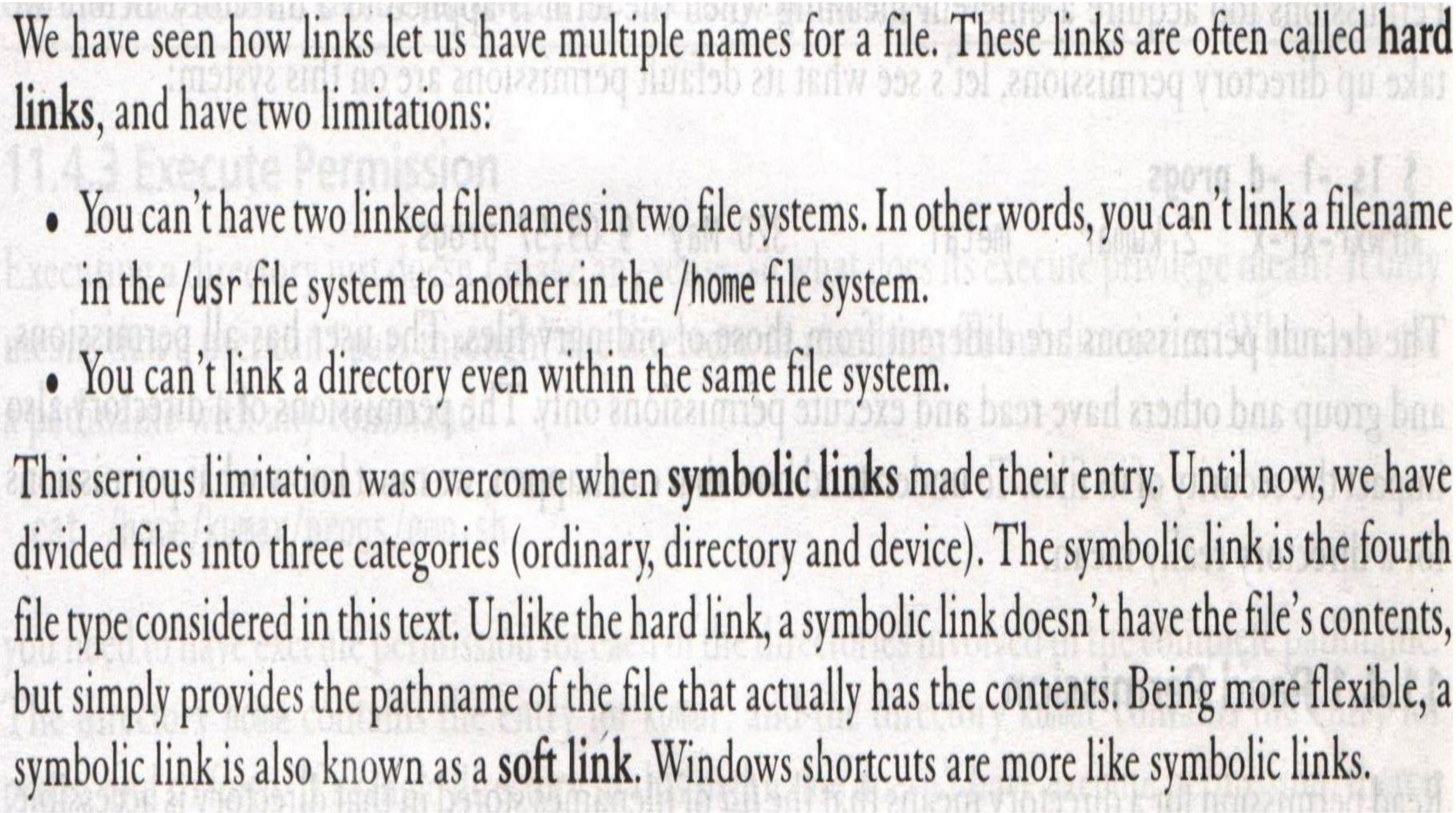
1. ln data/ foo.txt input_files
2. Links provide some protection against accidental deletion
3. Because of links, we don’t need to maintain two programs as two separate disk files.

LIMITATIONS OF HARD LINKS

1. We cant have two linked filenames in two file systems.
2. We cant link a directory even within the same file system

This can be solved by using symbolic links (soft links)

Symbolic Links

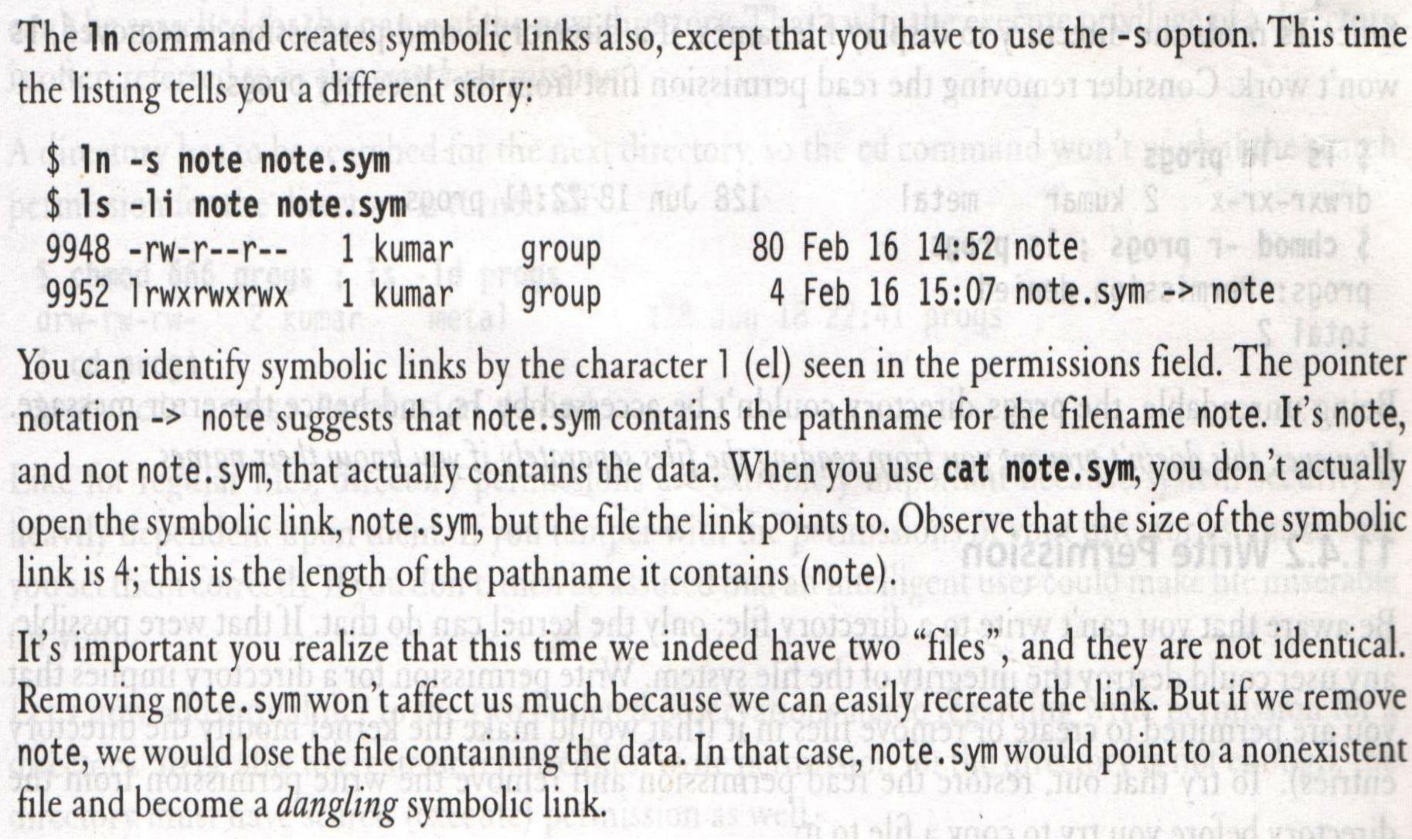


We have seen how links let us have multiple names for a file. These links are often called **hard links**, and have two limitations:

- You can't have two linked filenames in two file systems. In other words, you can't link a filename in the /usr file system to another in the /home file system.
- You can't link a directory even within the same file system.

This serious limitation was overcome when **symbolic links** made their entry. Until now, we have divided files into three categories (ordinary, directory and device). The symbolic link is the fourth file type considered in this text. Unlike the hard link, a symbolic link doesn't have the file's contents, but simply provides the pathname of the file that actually has the contents. Being more flexible, a symbolic link is also known as a **soft link**. Windows shortcuts are more like symbolic links.

Symbolic Links



The `ln` command creates symbolic links also, except that you have to use the `-s` option. This time the listing tells you a different story:

```
$ ln -s note note.sym
$ ls -li note note.sym
9948 -rw-r--r--   1 kumar    group          80 Feb 16 14:52 note
9952 lrwxrwxrwx   1 kumar    group           4 Feb 16 15:07 note.sym -> note
```

You can identify symbolic links by the character `l` (el) seen in the permissions field. The pointer notation `-> note` suggests that `note.sym` contains the pathname for the filename `note`. It's `note`, and not `note.sym`, that actually contains the data. When you use **`cat note.sym`**, you don't actually open the symbolic link, `note.sym`, but the file the link points to. Observe that the size of the symbolic link is 4; this is the length of the pathname it contains (`note`).

It's important you realize that this time we indeed have two "files", and they are not identical. Removing `note.sym` won't affect us much because we can easily recreate the link. But if we remove `note`, we would lose the file containing the data. In that case, `note.sym` would point to a nonexistent file and become a *dangling* symbolic link.

Thank you
Q & A

THANK YOU

BITS Pilani
Pilani Campus

COMPUTING AND DESIGN

SESSION 12

Course design by - Dr. Lucy J. Gudino &
Dr Chandrasekhar
Faculty – Dr Thangakumar J
WILP & Department of CS & IS

BITS Pilani
Pilani Campus

Operating Systems – File as a Logical Abstraction of Storage

BITS Pilani
Pilani Campus

Last Session

List of Topic Title	Text/Ref Book/external resource
- Levels in the file system - File storage on hard disks, CD ROMs and Flash ROMs - File Vs Databases	T1(16.1.2-16.1.3)

File as a Logical Abstraction of Storage

Today's topic

- File Access Methods – Effect of Locality
- File System Implementation
- Example (UNIX or Linux File System/ Windows NTFS/Mobile file System/ Flash Memory File System)

File Access Methods – Effect of Locality

- **Problem of old UNIX file system** - treated the disk like it was a random-access memory; data was spread all over the place so had a real and expensive positioning costs.
- Getting fragmented
- Smaller blocks were good because they minimized internal fragmentation, but bad for transfer as each block might require a positioning overhead to reach it.

Fast File System (FFS)

- To design the file system structures and allocation policies to be “disk aware”; improve performance.
- Keeping the same interface to the file system, but changing the internal implementation leads to performance and reliability.

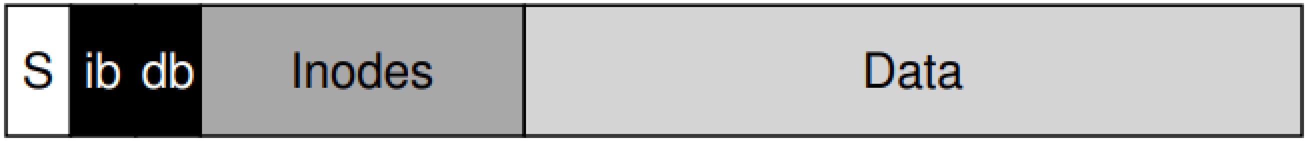
Organizing Structure: The Cylinder Group

- FFS divides the disk into a bunch of groups known as cylinder groups (block groups)

- FFS can ensure that accessing one after the other will not result in long seeks across the disk.

- It allocate files and directories within each of these groups.

- A per-group inode bitmap (ib) and data bitmap (db) serve this role for inodes and data blocks in each group.

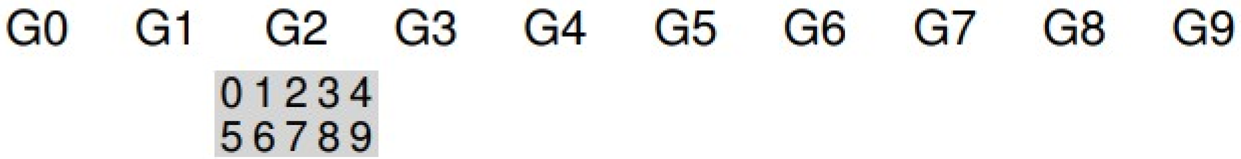


The Large-File Exception

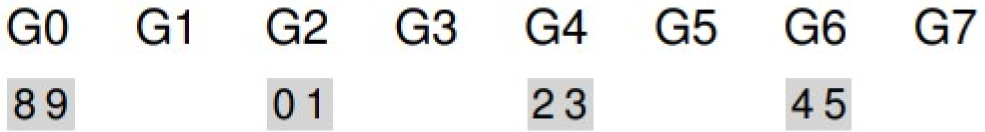
- File would entirely fill the block group it is first placed within.

- Filling a block group is undesirable, as it prevents subsequent “related” files from being placed within this block group, and thus may hurt file-access locality.

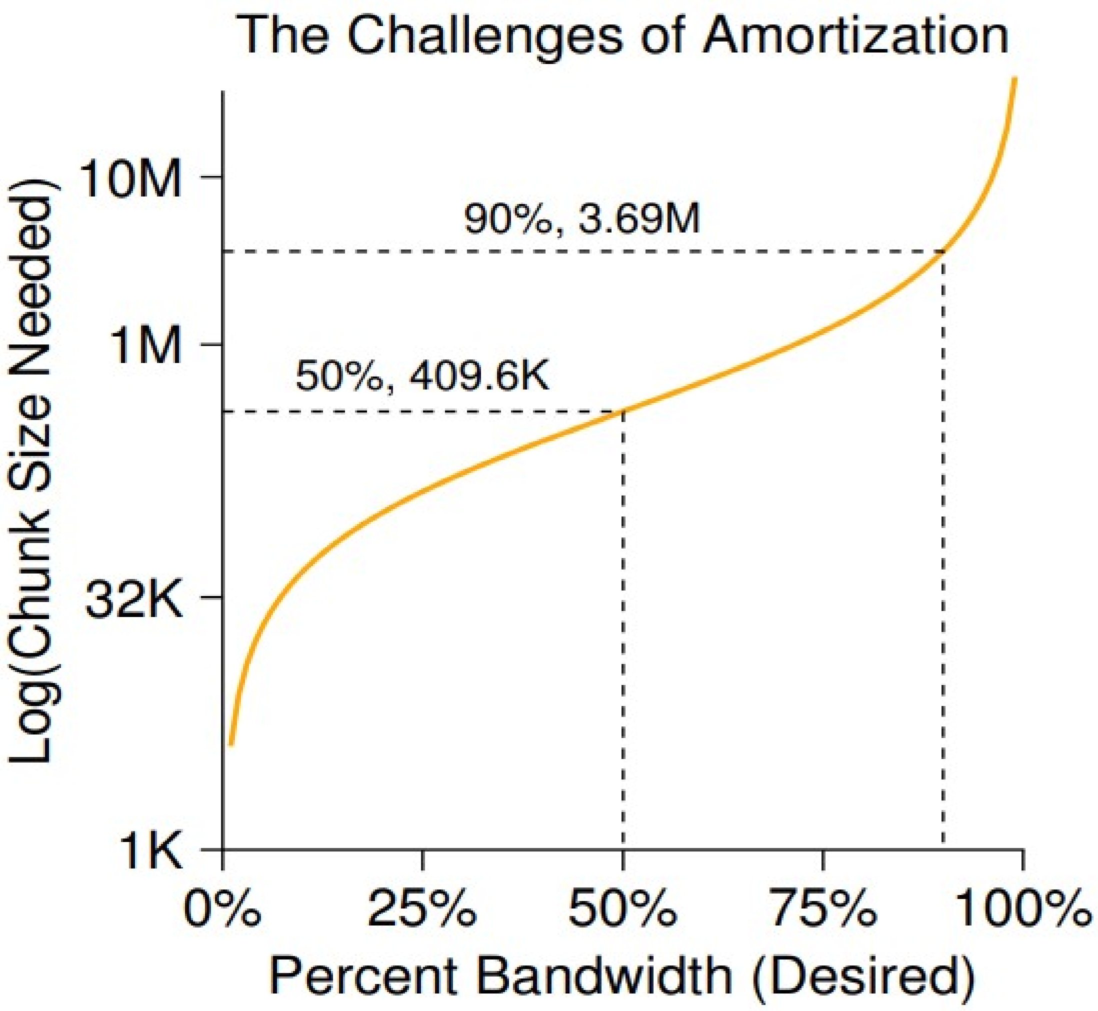
- The depiction of FFS without the large-file exception:


G0
G1
G2
G3
G4
G5
G6
G7
G8
G9
0 1 2 3 4
5 6 7 8 9

- With the large-file exception, the file spread across the disk in chunks:


G0
G1
G2
G3
G4
G5
G6
G7
8 9
0 1
2 3
4 5

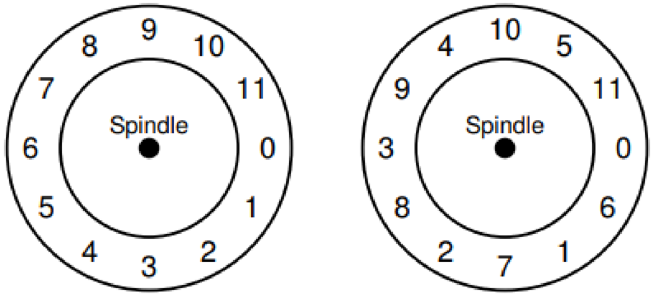
Amortization: How Big Do Chunks Have To Be?



- Many files were 2KB or so in size back then, and using 4KB blocks, while good for transferring data, was not so good for space efficiency.
- This internal fragmentation leads to roughly half the disk being wasted for a typical file system.
- Introduced sub-blocks, which were 512-byte little blocks that the file system could allocate to files
- FFS will find a 4KB block, copy the sub-blocks into it, and free the sub-blocks for future use.

FFS: Standard Versus Parameterized Placement

- **Parameterization** - FFS was smart enough to figure out for a particular disk how many blocks it should skip in doing layout to avoid the extra rotations
- Modern disks internally read the entire track in and buffer it in an internal disk cache – track buffer
- Symbolic links allow the user to create an “alias” to any other file or directory on a system and thus are much more flexible.
- FFS also introduced an atomic rename() operation for renaming files.



FILE SYSTEM IMPLEMENTATION

- To implement the file system, there is a need to implement three objects the file system provides for users:
    - **Files**
    - **Open Files**
    - **Directories**
- Each one is implemented as a data structure and a set of procedures to manipulate the data structure.

File Descriptor

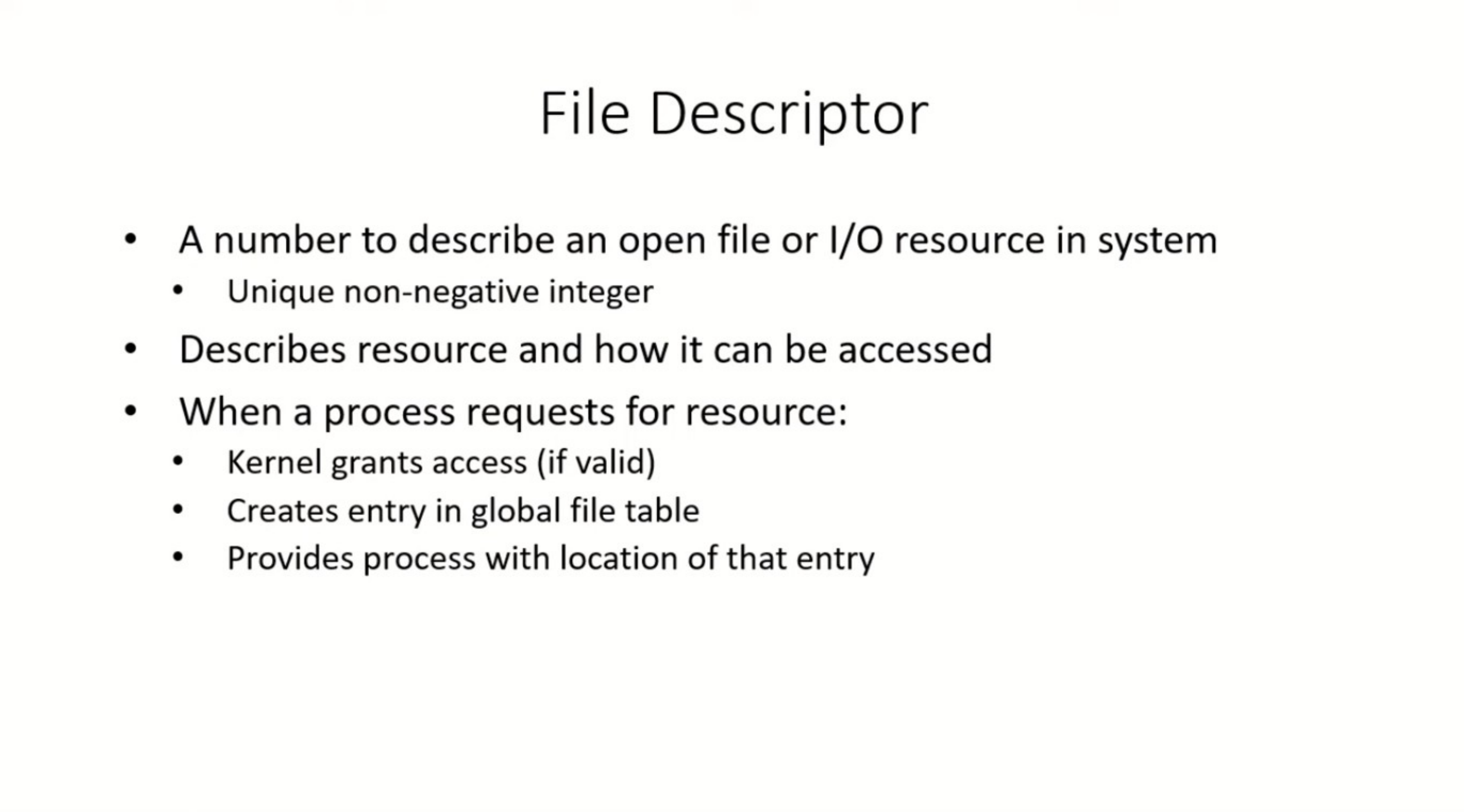
- **File descriptor** is simply an index into the file descriptor table. For each process in our operating system, there is a **process control block**(PCB). PCB keeps track of the context of the process. So one of the fields within this is an array called file descriptor table.

- This array keeps track of all the resources that the process owns and can operate on. The file descriptor table holds pointers to the resources. Resources could be
  - File
  - Terminal I/O
  - Pipes
  - Socket for the communication channel between machines
  - Any devices

- So if any process opens or uses any resources will have an entry in the file descriptor table. Any process when it first starts is given access to three resources.
- These resources are:
- stdin
- stdout
- stderr

- Suppose we are working on the terminal then input for it is stdin, the output is stdout and the error is stderr.
- And if we open any other terminal then both are two different processes now. The newly opened terminal will have its own stdin, stdout because it’s a different process.

File Descriptor



File Descriptor

- A number to describe an open file or I/O resource in system
    - Unique non-negative integer
- Describes resource and how it can be accessed
- When a process requests for resource:
    - Kernel grants access (if valid)
    - Creates entry in global file table
    - Provides process with location of that entry

- The file descriptor is just an integer that you get from the **open()** system call.
- Example of file descriptor:

int fd = open(filePath, mode);

File Pointer

- **File pointer** is a pointer returned by **fopen()** library function. It is used to identify a file. It is passed to a **fread()** and **fwrite()** function.
- Example of file pointer:

```
FILE *fp;
fp = fopen(“sample.txt,”a”);
fprintf( fp, “Welcome to GFG”);
fclose(fp);
```

File System Data Structures

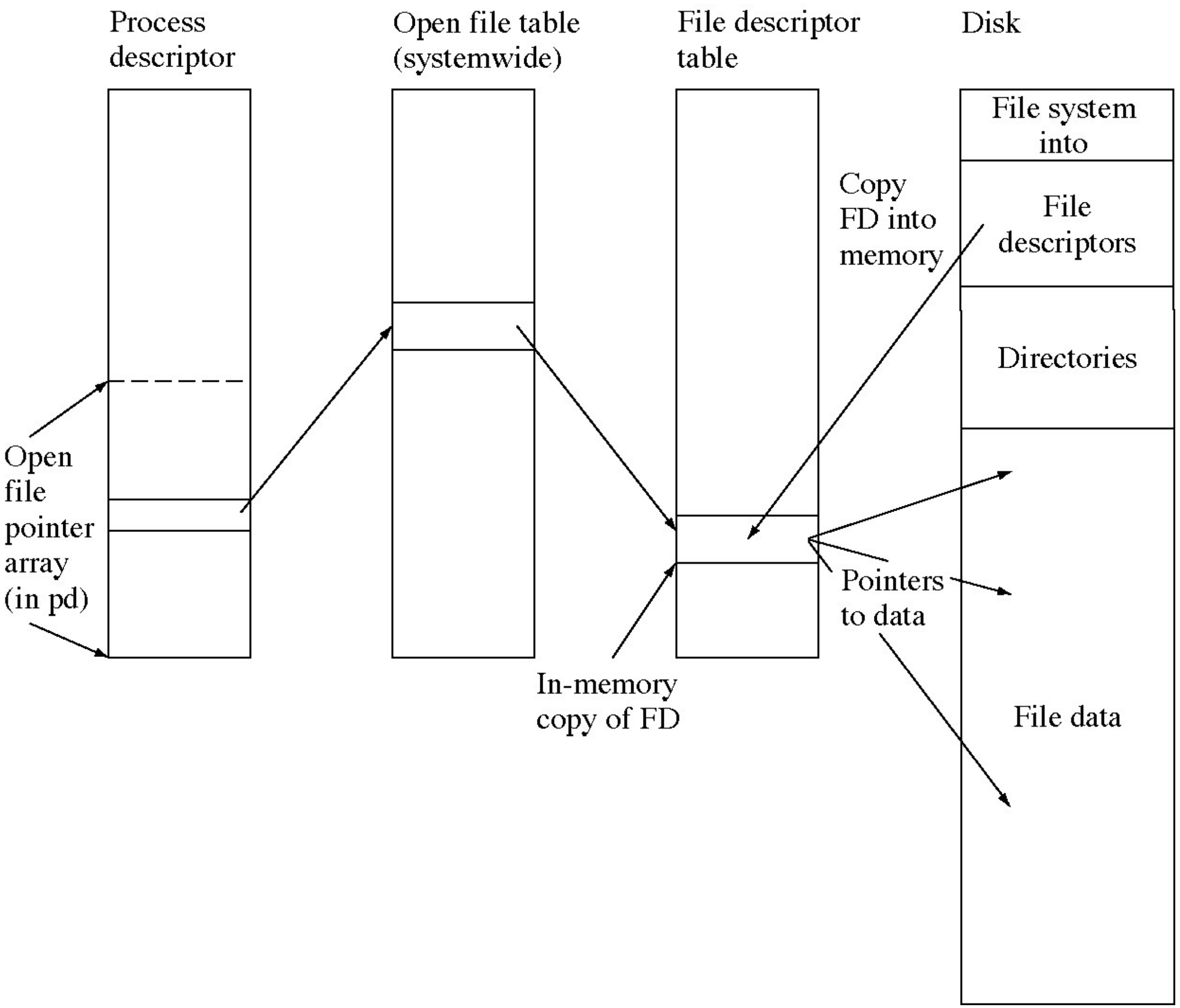
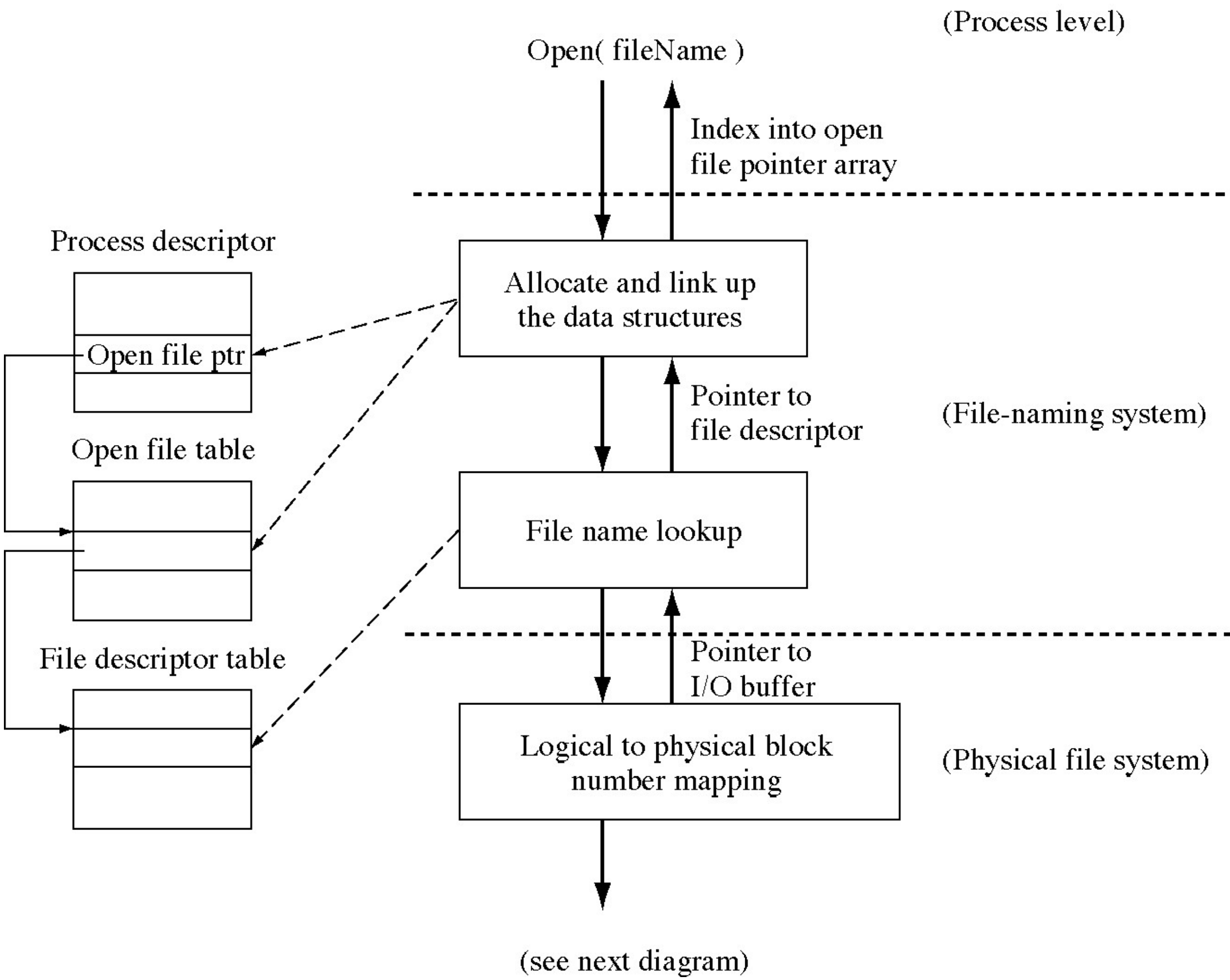

Process
descriptor
Open file table
(systemwide)
File descriptor
table
Disk
File system
into
File
descriptors
Directories
File data
Copy
FD into
memory
Open
file
pointer
array
(in pd)
Pointers
to data
In-memory
copy of FD

Figure shows the main data structures in a file system and how they are connected to each other. The “open file table” contains a structure for each open file. A file is opened by a process and is connected with that process. **Open File Pointer Table (or per process open file table)** - When a file is opened, the open file structure is allocated and a pointer to that structure is place in a tale in the process descriptor of the process opening the file.

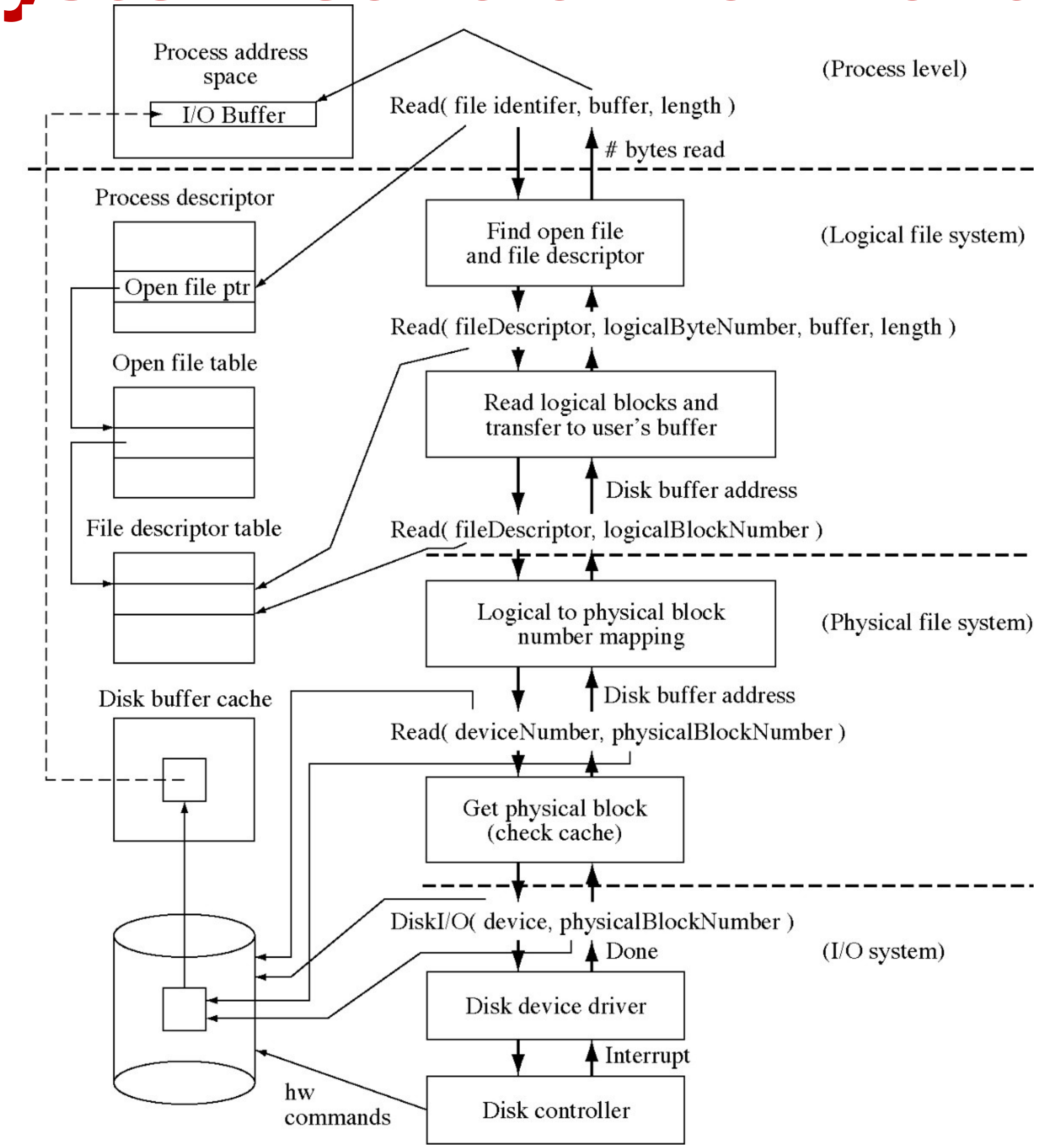
- **File Descriptor** – the data structure that represents the file (contains all the meta-data about the file).

File System Control Flow for an Open


(Process level)
Open(fileName)
Index into open
file pointer array
Process descriptor
Open file ptr
Allocate and link up
the data structures
Pointer to
file descriptor
(File-naming system)
Open file table
File name lookup
File descriptor table
Pointer to
I/O buffer
Logical to physical block
number mapping
(Physical file system)
(see next diagram)

- Figure shows the control flow for the **open** system call. **The left side** – data structure affected;
- **right side** – file system modules that are called.
- **The vertical arrows –** connects the modules represent procedure calls and returns.
- **The dashed arrows** – data structures the modules access and modify.

File System Control Flow for a Read



Process address space

I/O Buffer

(Process level)

Read(file identifer, buffer, length)

bytes read

Process descriptor

Open file ptr

Find open file and file descriptor

(Logical file system)

Read(fileDescriptor, logicalByteNumber, buffer, length)

Open file table

Read logical blocks and transfer to user's buffer

Disk buffer address

File descriptor table

Read(fileDescriptor, logicalBlockNumber)

Logical to physical block number mapping

(Physical file system)

Disk buffer cache

Disk buffer address

Read(deviceNumber, physicalBlockNumber)

Get physical block (check cache)

DiskI/O(device, physicalBlockNumber)

Done

(I/O system)

Disk device driver

Interrupt

hw commands

Disk controller

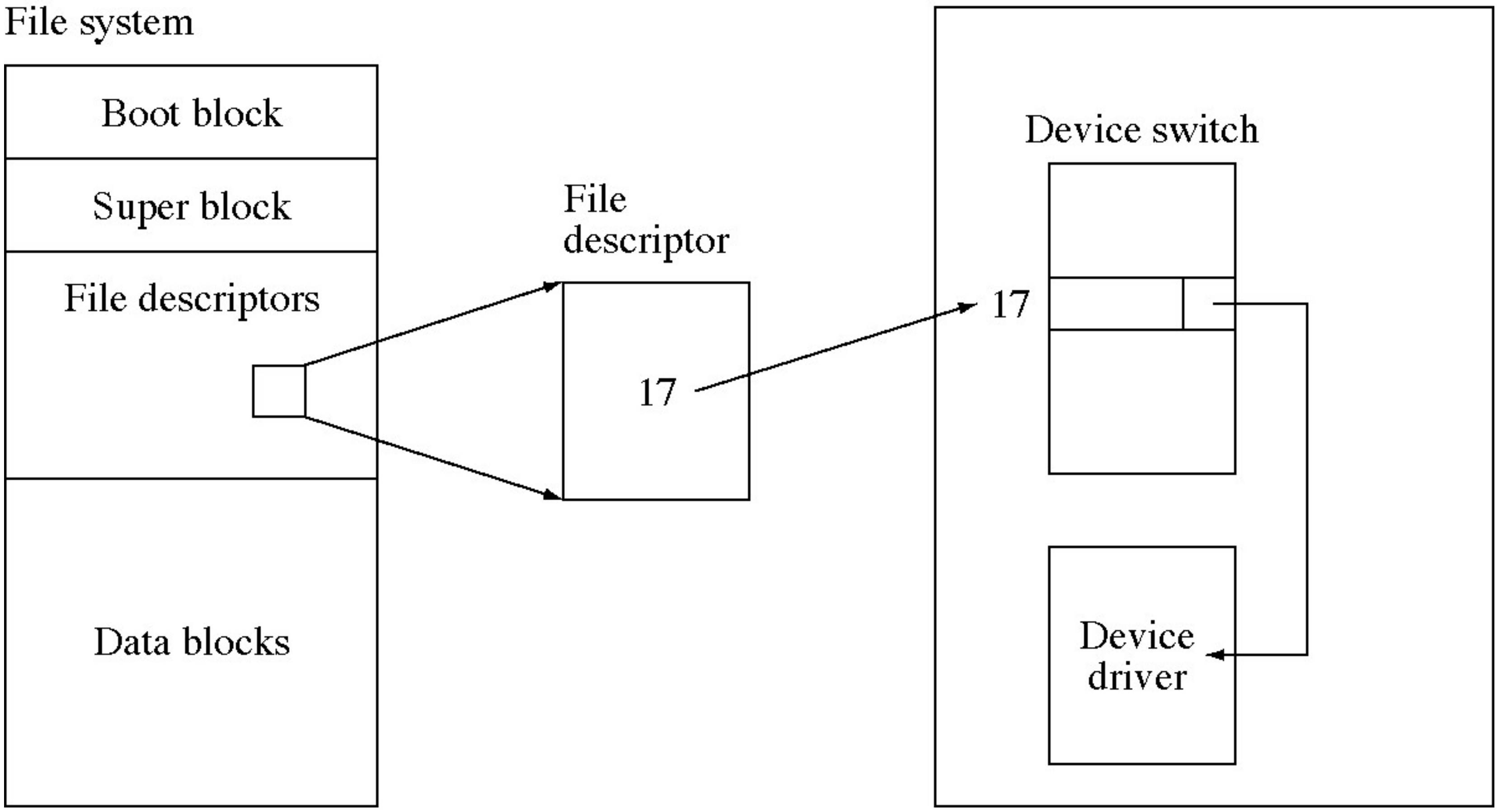
Figure shows the control flow for the **read** system call. **The left side** – the data structures and how they linked together.

The right side – the file system modules that use these data structures.

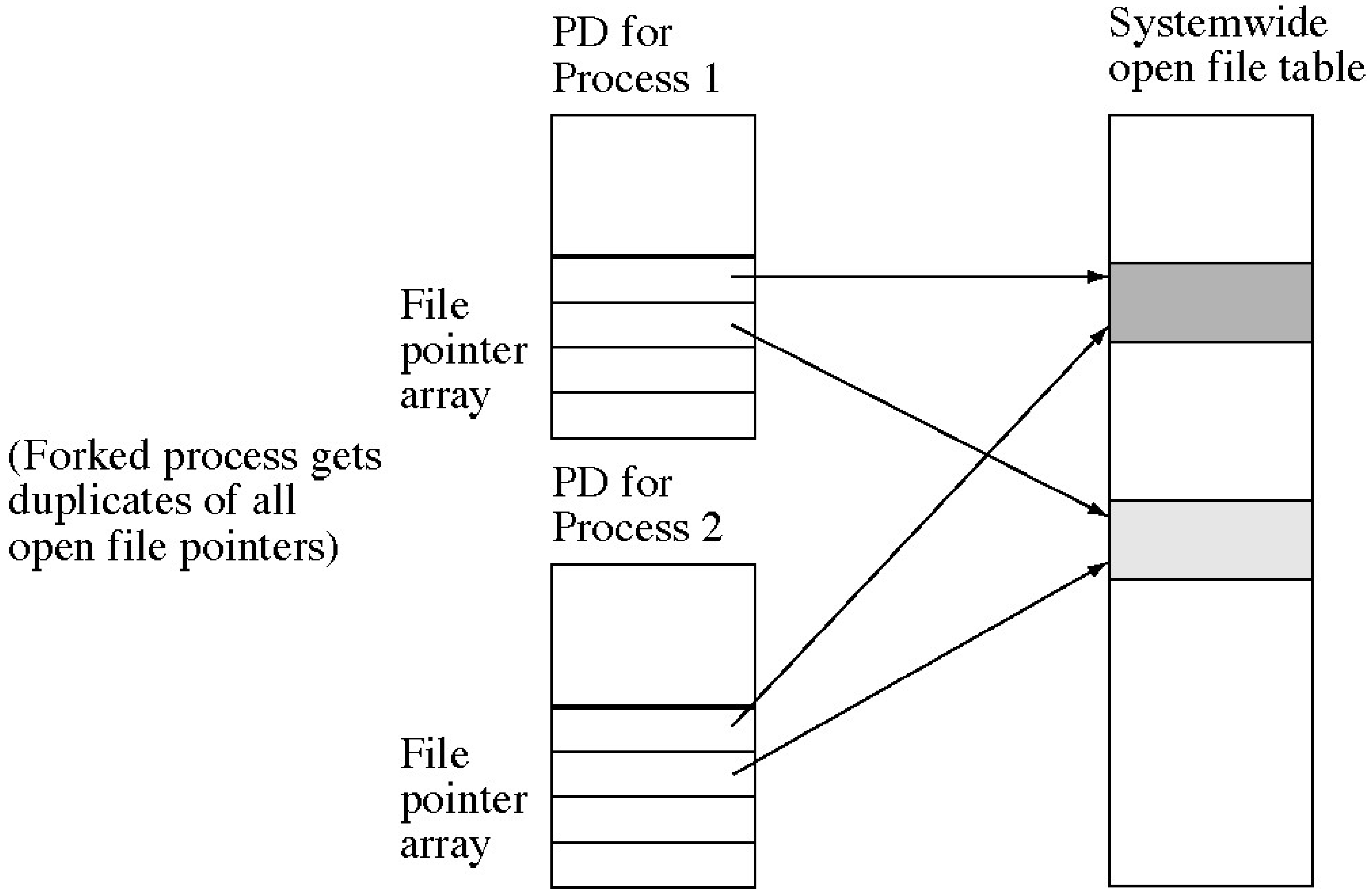
The downward arrows – procedure calls. **The upward arrows** – represents the return from the procedure call.

The lines from right side to the left side – indicate which data structures are accessed and modified by the various modules. **The logical file system** – files at the logical level (logical byte offsets, logical blocks, and logical block numbers). **The physical file system** - physical blocks. **The I/O system** – devices.

Connecting Files and Devices through the Device Switch


File system
Boot block
Super block
File descriptors
Data blocks
File descriptor
17
Device switch
17
Device driver

The Effect of a Fork on the File System Data Structure


PD for Process 1
Systemwide open file table
File pointer array
(Forked process gets duplicates of all open file pointers)
PD for Process 2
File pointer array

System Call Data Structure Changes

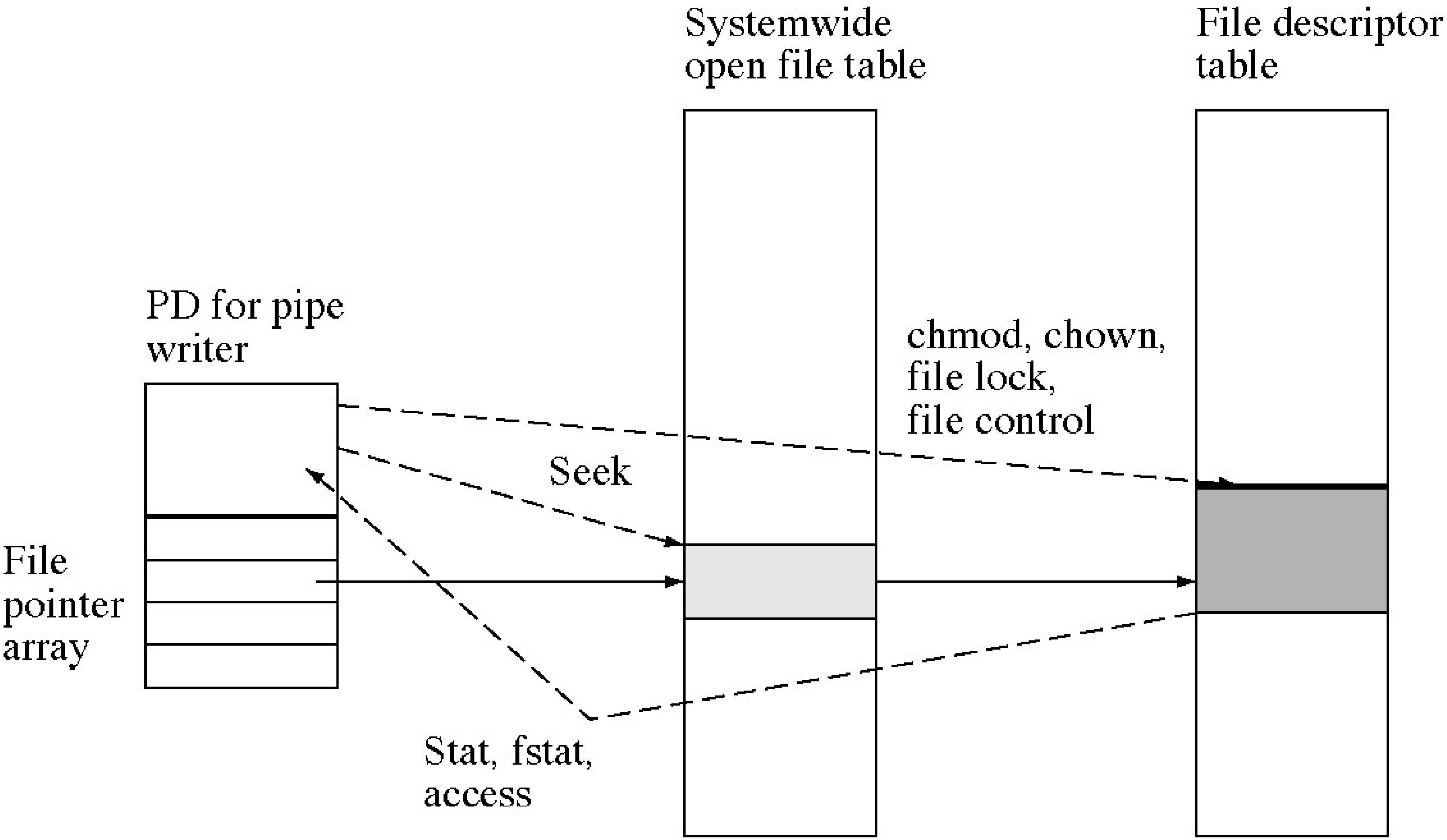

Systemwide
open file table
File descriptor
table
PD for pipe
writer
chmod, chown,
file lock,
file control
Seek
File
pointer
array
Stat, fstat,
access

Figure shows the effect of **stat, fstat, access, chmod, chown, and seek** system calls on the file system data structure.

The stat an fstat – find the file descriptor and return the meta-data in it.

The access system – finds the file descriptor and checks the protection information.

The chmod and chown system call – change the meta data in the file descriptor and arrange for it to be written back out to the disk.

The seek system call – changes the file pointer in the open file structure.

The file control and file lock system calls - change the information in the file descriptor.

Stat command

stat is a command which gives information about the file and filesystem. Stat command gives information such as the size of the file, access permissions and the user ID and group ID, birth time access time of the file.

syntax of using stat command:

```
stat --options filenames
```

Stat command

-L, --dereference follow links

-f, --file-system display file system status instead of file status

-c --format=FORMAT use the specified FORMAT instead of the default; output a newline after each use of FORMAT

Information we get from stat

- **File:** The file name, If the provided file is a symlink, then the name will be different.
- **Size:** The size of a given file in Bytes.
- **Blocks:** Total number of allocated blocks to the file on hard disk.
- **File type:** Regular files, special files, directories, or symbolic links.
- **Device:** Device number in hexadecimal format.
- **Inode:** Inode number of the file.
- **Links:** Number of hard links of the file.
- **Access:** The last time at which the file was accessed.
- **Modify:** The last time at which file was modified.
- **Change:** The last time the at which file’s attribute or content was changed.
- **Birth:** The time at which the file was created.

du Command

du command displays the number of blocks used for files. If the *File* parameter specified is actually a directory, all files within the directory are reported on. If no *File* parameter is provided, the **du** command uses the files in the current directory.

Notes:

- Files with multiple links are counted and written for only one entry.
- Block counts are based only on file size

du Command

Flags

-a Displays disk usage for each file specified, or displays the individual disk usage for each file in a directory.

-k Calculates the block count in 1024-byte units rather than the default 512-byte units.

-l Allocates blocks evenly among the links for files with multiple links. By default, a file with two or more links is counted only once.

-s Displays the total disk usage for all specified files, or displays the total disk usage for all files in a directory.

syntax of du command

du [OPTION]... [FILE]...

df Command

- df command (short for disk filesystem) is used to show disk utilization for a Linux system.
- **Display information of device name, total blocks, total disk space, used disk space, available disk space and mount points on a file system**
- Display all the file system, use -a option. df –a
- Use -h option to display size in power of 1024 df -h /home/HCL
- Use -H option to display sizes in power of 1000 df -H /home/HCL
- Use -T option to display file type df -T /home/HCL

df Command

To display information about /boot file system,

```
df /boot
```

We can display information in various block format. For instance see 1M-byte blocks, run:

```
df -m
df -m /boot
```

Show statistics about the number of free and used inodes

Pass the -i as follows to display index node (inodes) info:

```
df -i
df -i -H
df -i -H -T
```

Inodes

Each file is represented by a structure, called an inode.

Each inode contains the description of the file: file type, access rights, owners, timestamps, size, pointers to data blocks.

The addresses of data blocks allocated to a file are stored in its inode. When a user requests an I/O operation on the file, the kernel code converts the current offset to a block number, uses this number as an index in the block addresses table and reads or writes the physical block.

Directories

- Directories are structured in a hierarchical tree. Each directory can contain files and subdirectories.

- Directories are implemented as a special type of files. Actually, a directory is a file containing a list of entries. Each entry contains an inode number and a file name. When a process uses a pathname, the kernel code searchs in the directories to find the corresponding inode number. After the name has been converted to an inode number, the inode is loaded into memory and is used by subsequent requests.

Links

The inode contains a field containing the number associated with the file.

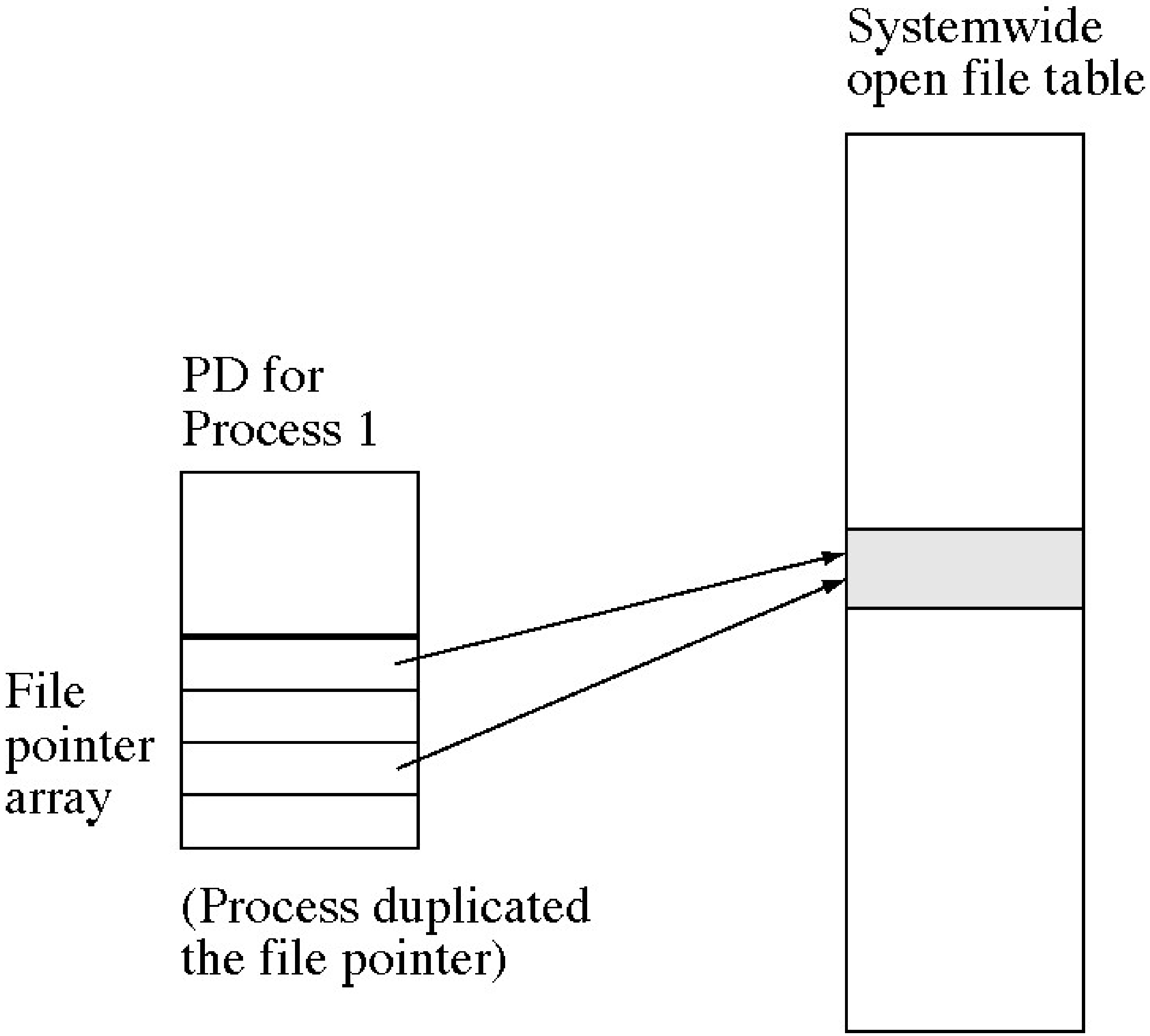
Adding a link simply consists in creating a directory entry, where the inode number points to the inode, and in incrementing the links count in the inode.

When a link is deleted, i.e. when one uses the rm command to remove a filename, the kernel decrements the links count and deallocates the inode if this count becomes zero.

Device special files

- Devices can be accessed via special files. A device special file does not use any space on the filesystem. It is only an access point to the device driver.
- Two types of special files exist: character and block special files. The former allows I/O operations in character mode while the later requires data to be written in block mode via the buffer cache functions.
- When an I/O request is made on a special file, it is forwarded to a device driver. A special file is referenced by a major number, which identifies the device type, and a minor number, which identifies the unit.

Effects of Duplicate System Call



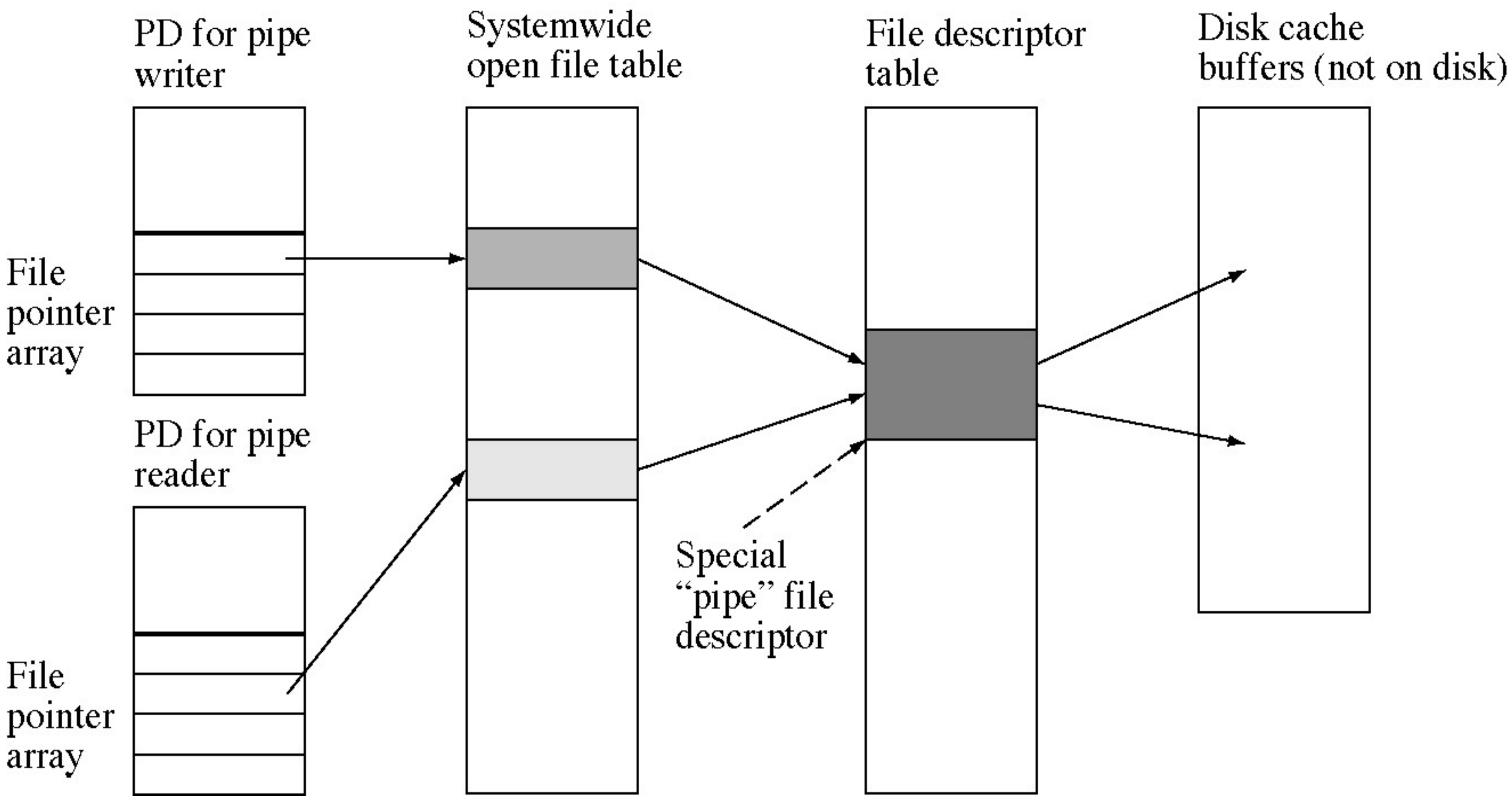
Systemwide
open file table

PD for
Process 1

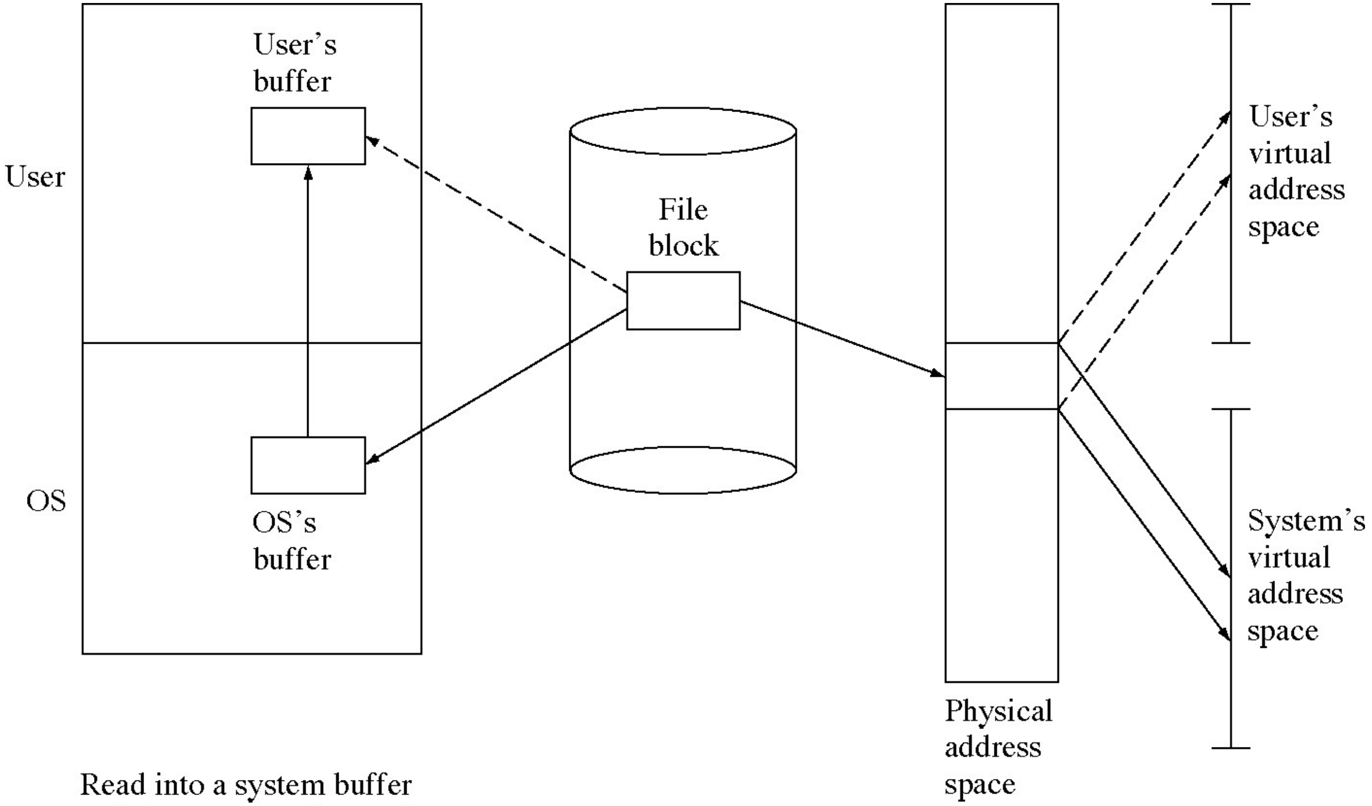
File
pointer
array

(Process duplicated
the file pointer)

Effects of Pipe System Call


PD for pipe writer
Systemwide open file table
File descriptor table
Disk cache buffers (not on disk)
File pointer array
PD for pipe reader
Special "pipe" file descriptor
File pointer array

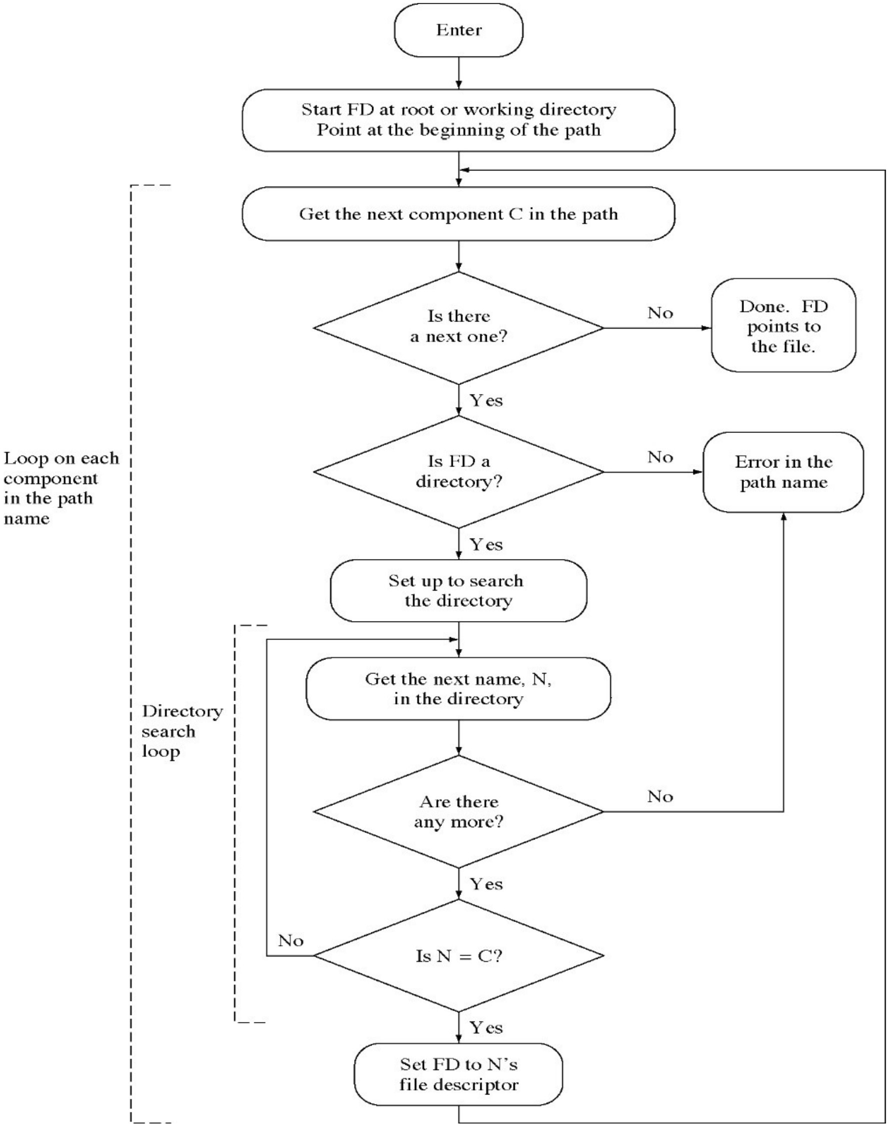
Avoiding Copying I/O Data


User's buffer
User
File block
OS
OS's buffer
User's virtual address space
System's virtual address space
Physical address space

Read into a system buffer and then copy to the user's buffer, or read directly into the user's buffer.

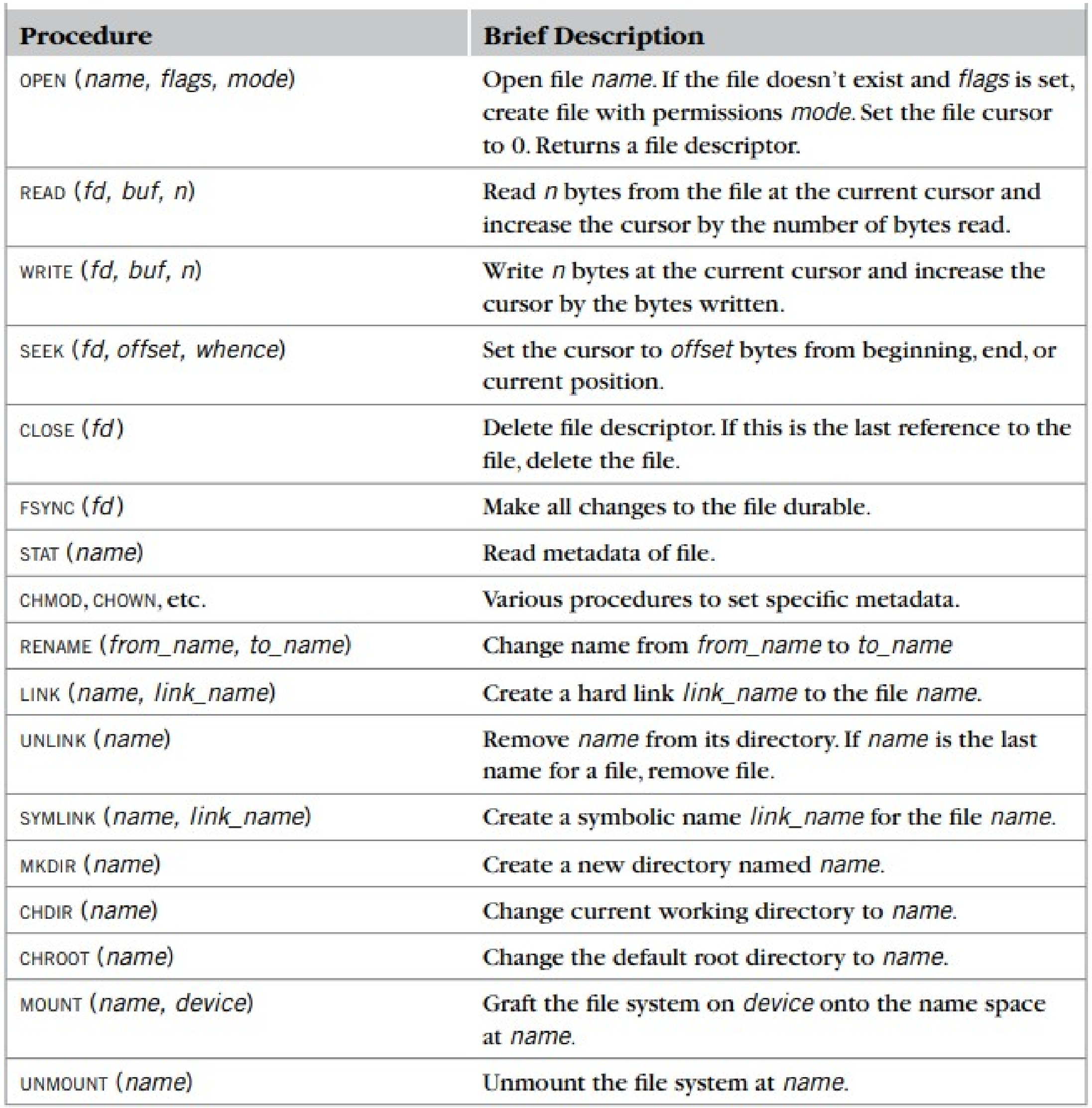
Copy into a page in the system's virtual address space and then map it into the user's virtual address space.

Path Name Lookup Algorithm


Enter
Start FD at root or working directory
Point at the beginning of the path
Get the next component C in the path
Is there a next one?
No
Done. FD points to the file.
Yes
Is FD a directory?
No
Error in the path name
Yes
Set up to search the directory
Get the next name, N, in the directory
Are there any more?
No
Yes
Is N = C?
No
Yes
Set FD to N's file descriptor
Loop on each component in the path name
Directory search loop

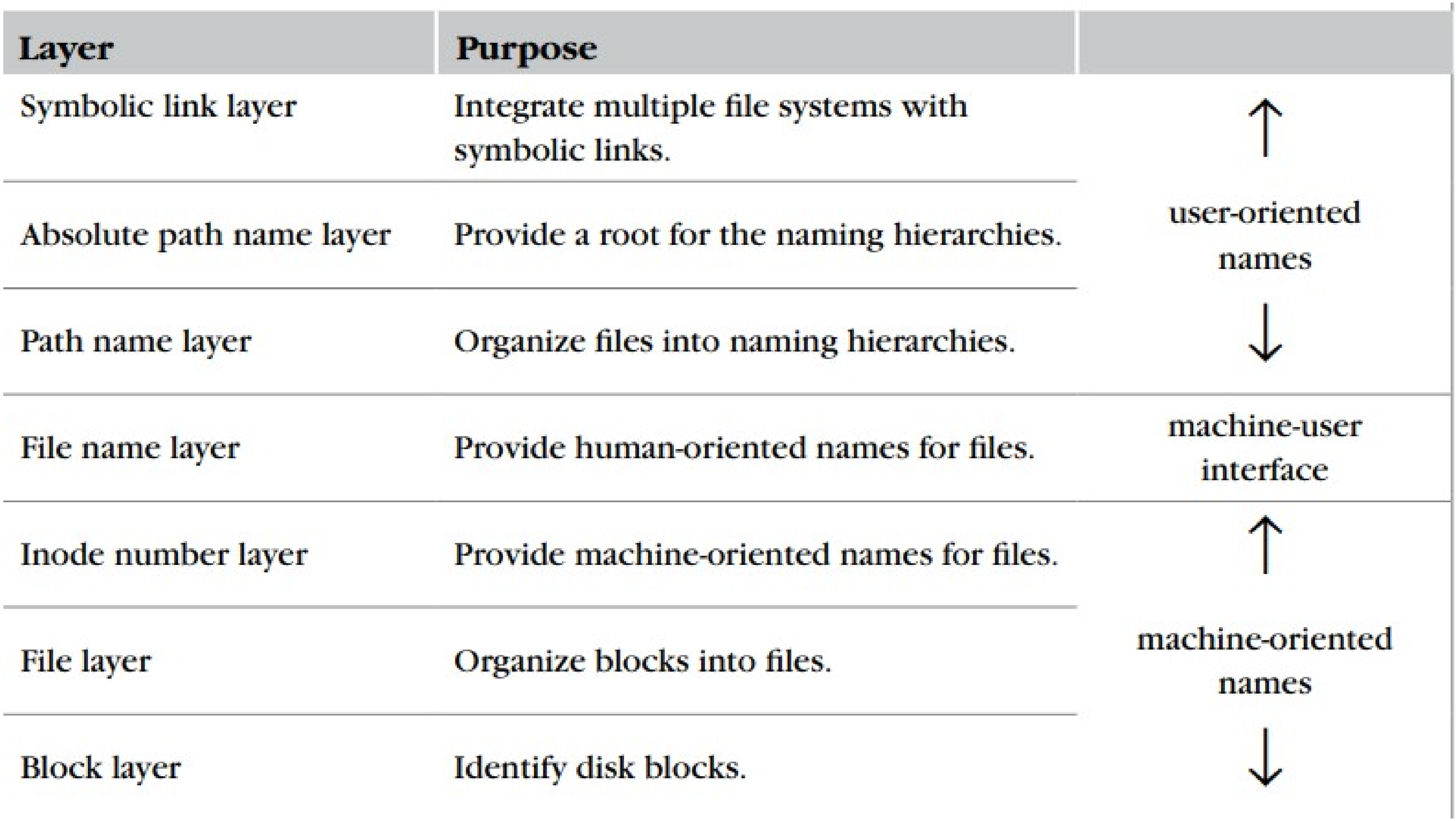
UNIX FILE SYSTEM

- Developed by Bell Telephone Laboratories for the Digital Equipment Corporation PDP line of minicomputers
- Most of the UNIX interface is an official standard



Procedure	**Brief Description**
OPEN (*name, flags, mode*)	Open file *name*. If the file doesn't exist and *flags* is set, create file with permissions *mode*. Set the file cursor to 0. Returns a file descriptor.
READ (*fd, buf, n*)	Read *n* bytes from the file at the current cursor and increase the cursor by the number of bytes read.
WRITE (*fd, buf, n*)	Write *n* bytes at the current cursor and increase the cursor by the bytes written.
SEEK (*fd, offset, whence*)	Set the cursor to *offset* bytes from beginning, end, or current position.
CLOSE (*fd*)	Delete file descriptor. If this is the last reference to the file, delete the file.
FSYNC (*fd*)	Make all changes to the file durable.
STAT (*name*)	Read metadata of file.
CHMOD, CHOWN, etc.	Various procedures to set specific metadata.
RENAME (*from_name, to_name*)	Change name from *from_name* to *to_name*
LINK (*name, link_name*)	Create a hard link *link_name* to the file *name*.
UNLINK (*name*)	Remove *name* from its directory. If *name* is the last name for a file, remove file.
SYMLINK (*name, link_name*)	Create a symbolic name *link_name* for the file *name*.
MKDIR (*name*)	Create a new directory named *name*.
CHDIR (*name*)	Change current working directory to *name*.
CHROOT (*name*)	Change the default root directory to *name*.
MOUNT (*name, device*)	Graft the file system on *device* onto the name space at *name*.
UNMOUNT (*name*)	Unmount the file system at *name*.

The Naming Layers of the UNIX File System



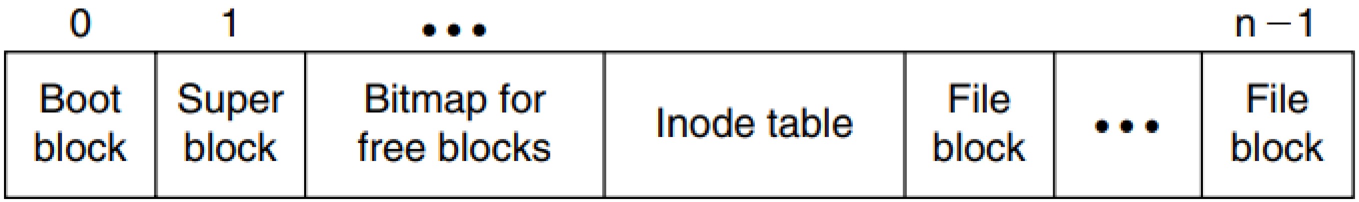
Layer	Purpose	
Symbolic link layer	Integrate multiple file systems with symbolic links.	↑ user-oriented names ↓
Absolute path name layer	Provide a root for the naming hierarchies.	
Path name layer	Organize files into naming hierarchies.	
File name layer	Provide human-oriented names for files.	machine-user interface
Inode number layer	Provide machine-oriented names for files.	↑ machine-oriented names ↓
File layer	Organize blocks into files.	
Block layer	Identify disk blocks.	

Disk layout for a simple file system

- Bottom layer the UNIX file system names some physical device (magnetic disk, flash disk, or magnetic tape) that can store data durably.

- The storage is divided into fixed-size units, called **blocks**.

- In **version 6,** the UNIX file system used 512-byte blocks, but modern UNIX file systems often use 8-kilobyte blocks.


0
1
• • •
n – 1
Boot block
Super block
Bitmap for free blocks
Inode table
File block
• • •
File block

- A file is a linear array of bytes of arbitrary length.

- The file system needs to record in some way which blocks belong to each file.

- To support this requirement, the UNIX file system creates an index node (**inode**).

- The inode for a file is a context in which the various blocks of the file are named by integer block numbers.

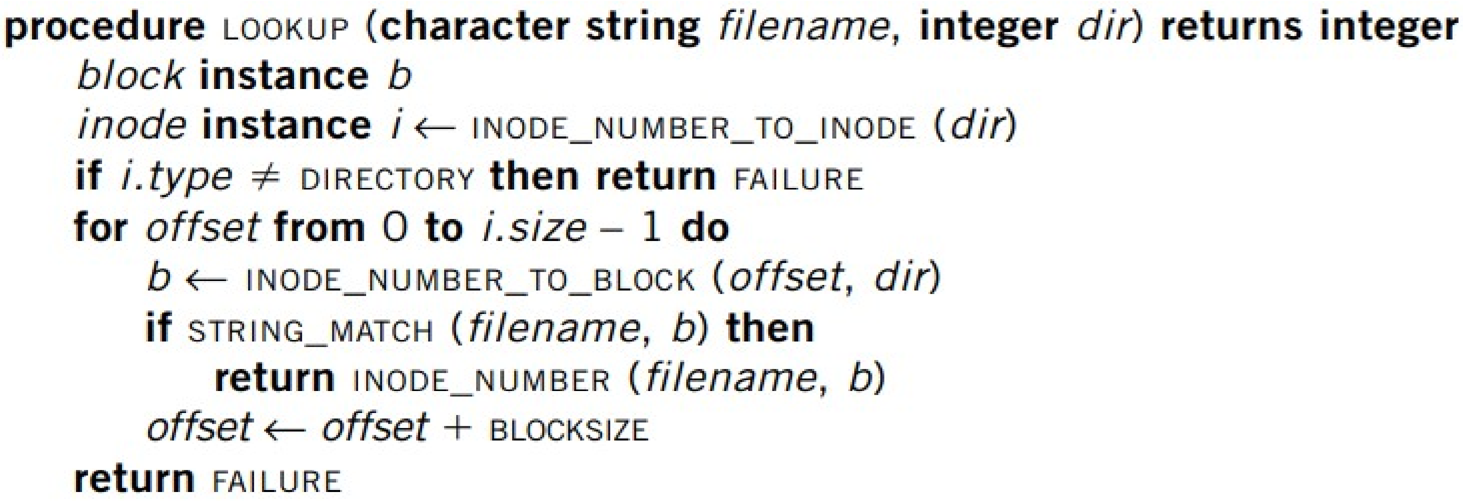
- To implement the naming layer is to employ a table that directly contains all inodes, indexed by inode number.

•**Directory** - The most visible component of the durable object naming scheme

•Figure is a directory with two file names ("program" and "paper"), which are mapped to inode numbers 10 and 12, respectively

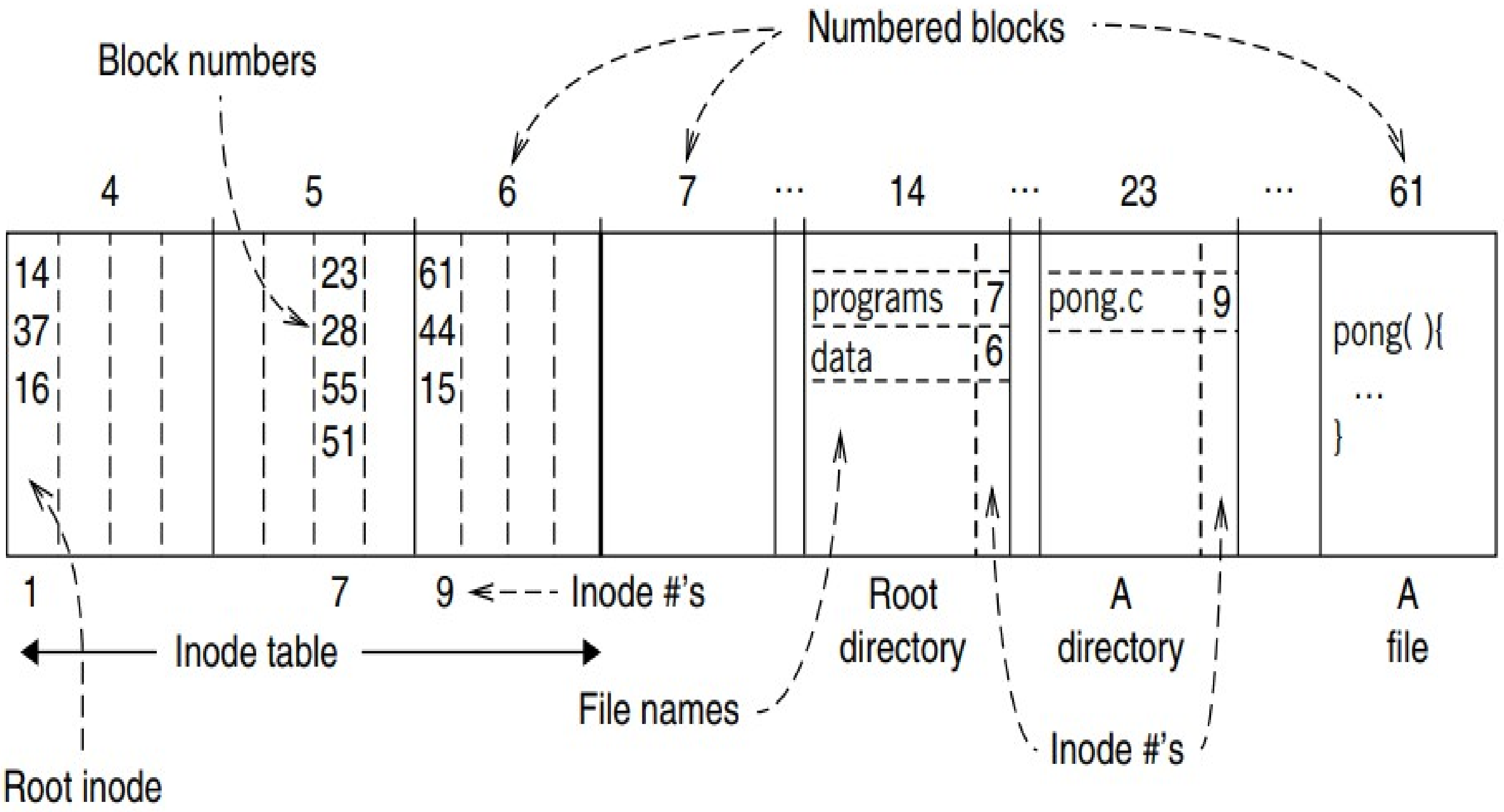
•In **Version 6**, the maximum length of a name is 14 bytes, and the entries in the table have a fixed length of 16 bytes

•\

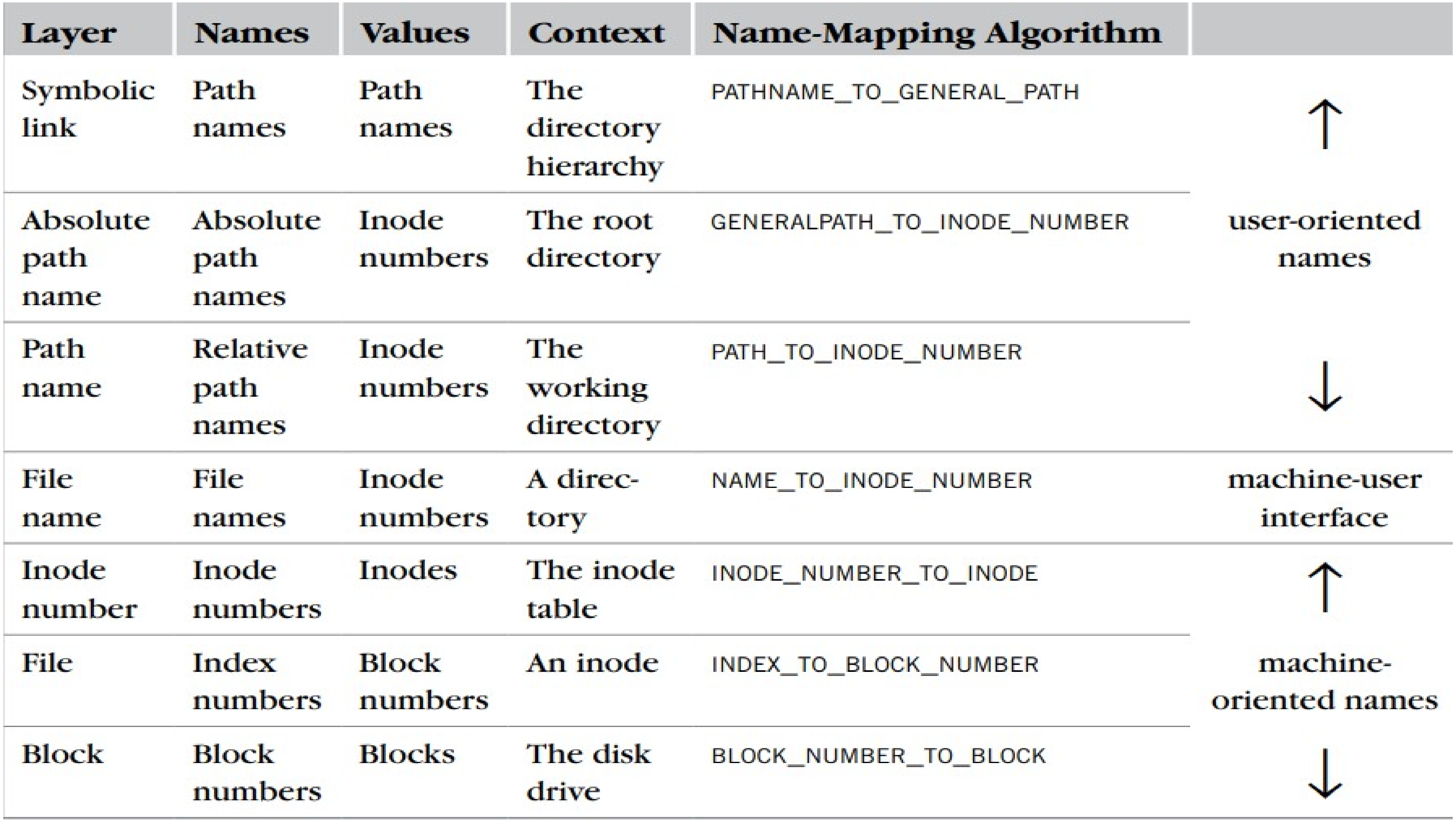


```
procedure LOOKUP (character string filename, integer dir) returns integer
    block instance b
    inode instance i ← INODE_NUMBER_TO_INODE (dir)
    if i.type ≠ DIRECTORY then return FAILURE
    for offset from 0 to i.size – 1 do
        b ← INODE_NUMBER_TO_BLOCK (offset, dir)
        if STRING_MATCH (filename, b) then
            return INODE_NUMBER (filename, b)
        offset ← offset + BLOCKSIZE
    return FAILURE
```

Example disk layout for a UNIX file system


Numbered blocks
Block numbers
4
5
6
7
…
14
…
23
…
61
14
37
16
23
28
55
51
61
44
15
programs
7
data
6
pong.c
9
pong( ){
…
}
1
7
9
Inode #'s
Root directory
A directory
A file
Inode table
File names
Inode #'s
Root inode

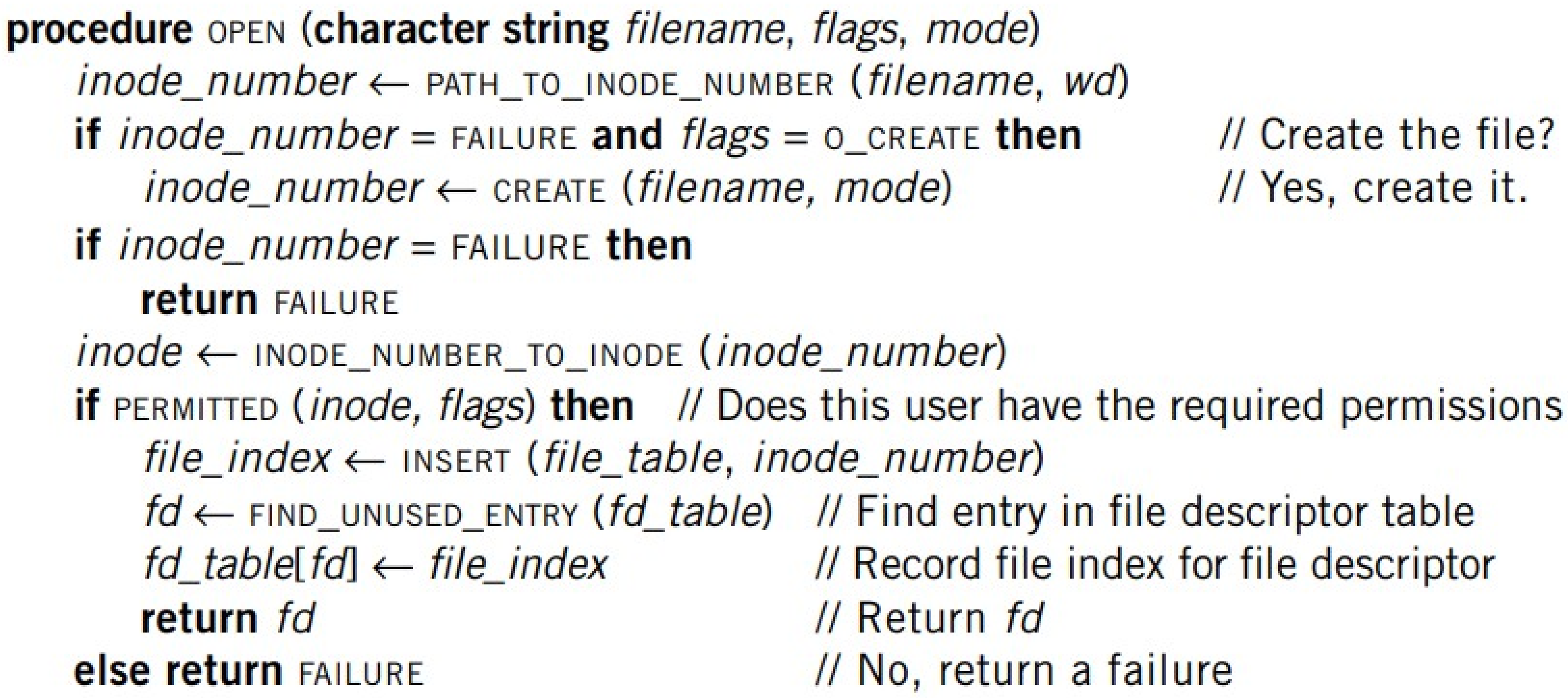
The UNIX Naming Layers, with Details of the Naming Scheme of each Layer



Layer	Names	Values	Context	Name-Mapping Algorithm	
Symbolic link	Path names	Path names	The directory hierarchy	PATHNAME_TO_GENERAL_PATH	↑
Absolute path name	Absolute path names	Inode numbers	The root directory	GENERALPATH_TO_INODE_NUMBER	user-oriented names
Path name	Relative path names	Inode numbers	The working directory	PATH_TO_INODE_NUMBER	↓
File name	File names	Inode numbers	A directory	NAME_TO_INODE_NUMBER	machine-user interface
Inode number	Inode numbers	Inodes	The inode table	INODE_NUMBER_TO_INODE	↑
File	Index numbers	Block numbers	An inode	INDEX_TO_BLOCK_NUMBER	machine-oriented names
Block	Block numbers	Blocks	The disk drive	BLOCK_NUMBER_TO_BLOCK	↓

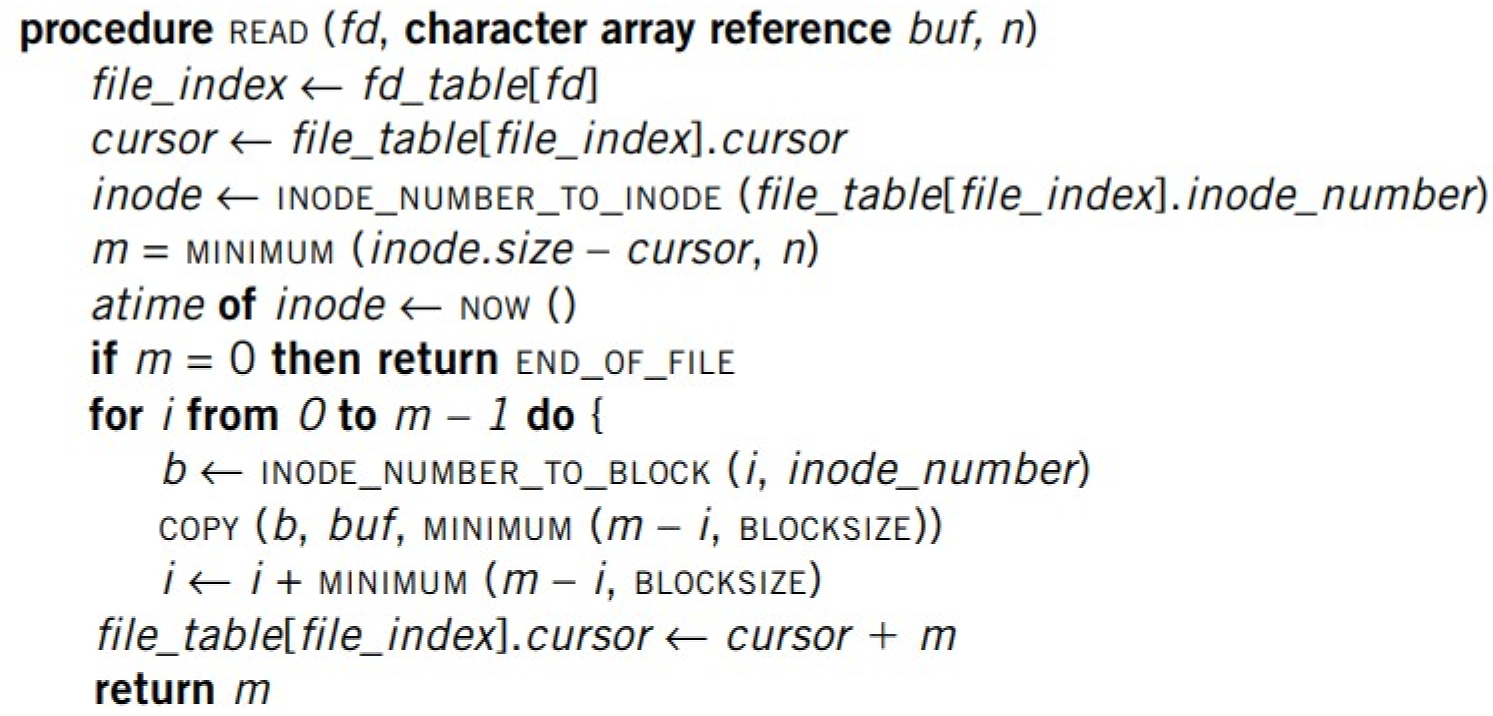
Implementing the File System API

- **File descriptor (fd) - READ, WRITE,** and **CLOSE** use to name the file



```
procedure OPEN (character string filename, flags, mode)
    inode_number ← PATH_TO_INODE_NUMBER (filename, wd)
    if inode_number = FAILURE and flags = O_CREATE then        // Create the file?
        inode_number ← CREATE (filename, mode)                 // Yes, create it.
    if inode_number = FAILURE then
        return FAILURE
    inode ← INODE_NUMBER_TO_INODE (inode_number)
    if PERMITTED (inode, flags) then    // Does this user have the required permissions
        file_index ← INSERT (file_table, inode_number)
        fd ← FIND_UNUSED_ENTRY (fd_table)    // Find entry in file descriptor table
        fd_table[fd] ← file_index            // Record file index for file descriptor
        return fd                            // Return fd
    else return FAILURE                      // No, return a failure
```

- **READ** is implemented as follows:



```
procedure READ (fd, character array reference buf, n)
    file_index ← fd_table[fd]
    cursor ← file_table[file_index].cursor
    inode ← INODE_NUMBER_TO_INODE (file_table[file_index].inode_number)
    m = MINIMUM (inode.size – cursor, n)
    atime of inode ← NOW ()
    if m = 0 then return END_OF_FILE
    for i from 0 to m – 1 do {
        b ← INODE_NUMBER_TO_BLOCK (i, inode_number)
        COPY (b, buf, MINIMUM (m – i, BLOCKSIZE))
        i ← i + MINIMUM (m – i, BLOCKSIZE)
    file_table[file_index].cursor ← cursor + m
    return m
```

REFERENCES

- **Data, S. I. Locality and The Fast File System.**

RECAP

List of Topic Title	Text/Ref Book/external resource

- I/O System Hardware
  - Device Controllers
  - Types of devices : Terminal Devices, Communication Devices and Disk Devices
- Data transfer schemes
- Programmed I/O, Interrupt Driven I/O, DMA
	T1 (14.1 – 14.4)
	R6 (7.3-7.5)

List of Topic Title	Text/Ref Book/external resource
• **I/O System Software** o Device Drivers o Device Driver Interfaces ▪ The Block Device Interfaces ▪ The Character Device Interfaces	T1 (15.1, 15.4)
o Device Numbers • **Unification of Files and I/O Devices** • **Mechanisms for improving I/O performance** o Prefetching o Caching o Buffering	T1 (15.5)

- Levels in the file system
- File storage on hard disks, CD ROMs and Flash ROMs
- File Vs Databases

THANK YOU

BITS Pilani
Pilani Campus

BITS Pilani
Pilani Campus

COMPUTING AND DESIGN

SESSION 13

Course design by - Dr. Lucy J. Gudino &
Dr Chandrasekhar
Faculty – Balamurali Shankar
WILP & Department of CS & IS

Operating Systems – Software Design

BITS Pilani
Pilani Campus

Last Session

List of Topic Title	Text/Ref Book/external resource
• File Access Methods – Effect of Locality • File System Implementation • Example (UNIX or Linux File System/ Windows NTFS/Mobile file System/ Flash Memory File System)	T2 and Class Notes

Today's topic

- **The General Concept of Abstraction**
- **Types of Abstractions in Software Engineering**
  - Data Abstraction
  - Control Abstraction
- **Design Styles in Programming and Programming Languages: Procedural Design vs. OO Design**
  - Object Oriented Abstraction
  - The Evolution of the Object Model
  - Foundations of the Object Model
  - Elements of the Object Model
  - Applying the Object Model
  - Abstractions in OOP

The General Concept of Abstraction

- **Abstraction -** a method to’ hide irrelevant details and represent only the essential features
- *An **abstraction** denotes the essential characteristics of an object that distinguish it from all other kinds of objects and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer*
- It allows managing complex systems
- Two types of abstraction:
  - **Data abstraction** - hiding the details about the data
  - **Control abstraction -** hiding the implementation details

```
#include <iostream>
using namespace std;
class implementAbstraction
{
        private:
                int a, b;

        public:

                // method to set values of
                // private members
                void set(int x, int y)
                {
                        a = x;
                        b = y;
                }

                void display()
                {
                        cout<<"a = " <<a << endl;
                        cout<<"b = " << b << endl;
                }
};

int main()
{
        implementAbstraction obj;
        obj.set(10, 20);
        obj.display();
        return 0;
}
```

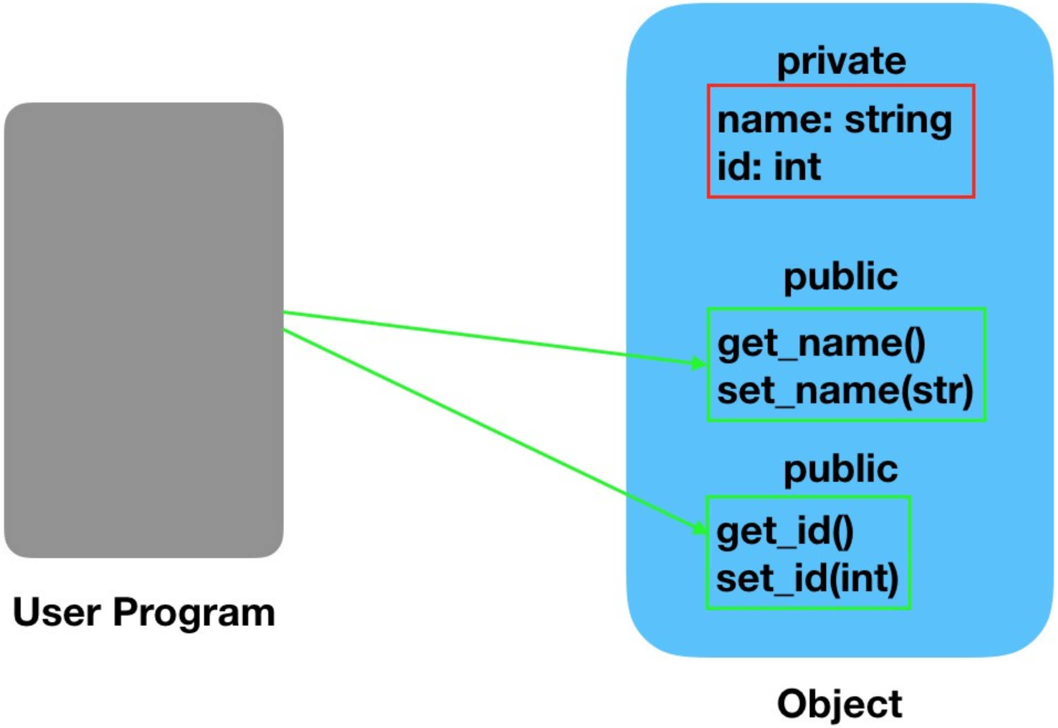
```
// Abstract class
abstract class Vehicle {
        // Abstract method (does not have a body)
        public abstract void VehicleSound();
        // Regular method
        public void honk() { System.out.println("honk honk"); }
}

// Subclass (inherit from Vehicle)
class Car extends Vehicle {
        public void VehicleSound()
        {
                // The body of VehicleSound() is provided here
                System.out.println("kon kon");
        }
}

class Main {
        public static void main(String[] args)
        {
                // Create a Car object
                Car myCar = new Car();
                myCar.VehicleSound();
                myCar.honk();
        }
}
```

Advantages of Abstraction

1. It reduces the complexity of viewing the things.
2. Avoids code duplication and increases reusability.
3. Helps to increase the security of an application or program as only important details are provided to the user.

Data Abstraction

Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.



private

name: string
id: int

public

get_name()
set_name(str)

public

get_id()
set_id(int)

User Program

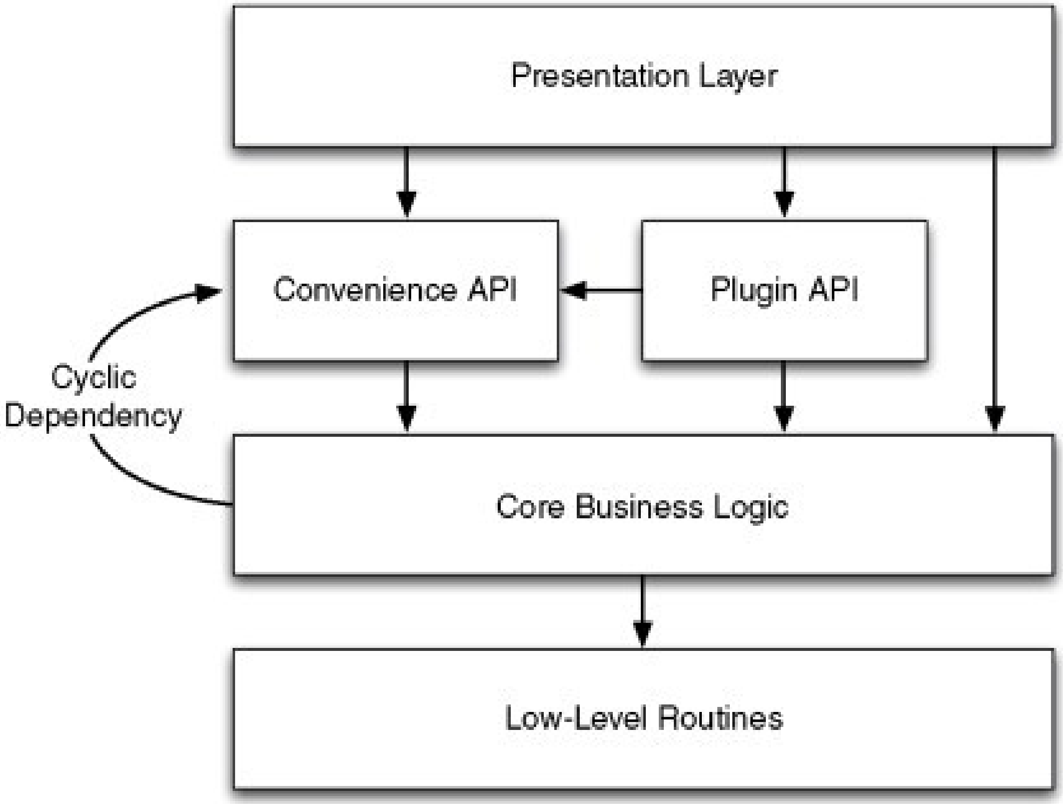
Object

Advantages of Data Abstraction

- Helps the user to avoid writing the low level code
- Avoids code duplication and increases reusability.
- Can change internal implementation of class independently without affecting the user.
- Helps to increase security of an application or program as only important details are provided to the user.

Control Abstraction

- This refers to the software part of abstraction wherein the program is simplified, and unnecessary execution details are removed.
- The **advantage** of control abstraction is that it makes code a lot cleaner and more secure.
- It is used to build new functionalities and combines control statements into a single unit
- Higher-order functions, closures, and lambdas are few preconditions for control abstraction.
- Highlights more on how a particular functionality can be achieved rather than describing each detail.
- Forms the main unit of structured programming.



Presentation Layer

Convenience API

Plugin API

Cyclic Dependency

Core Business Logic

Low-Level Routines

Procedural Programming

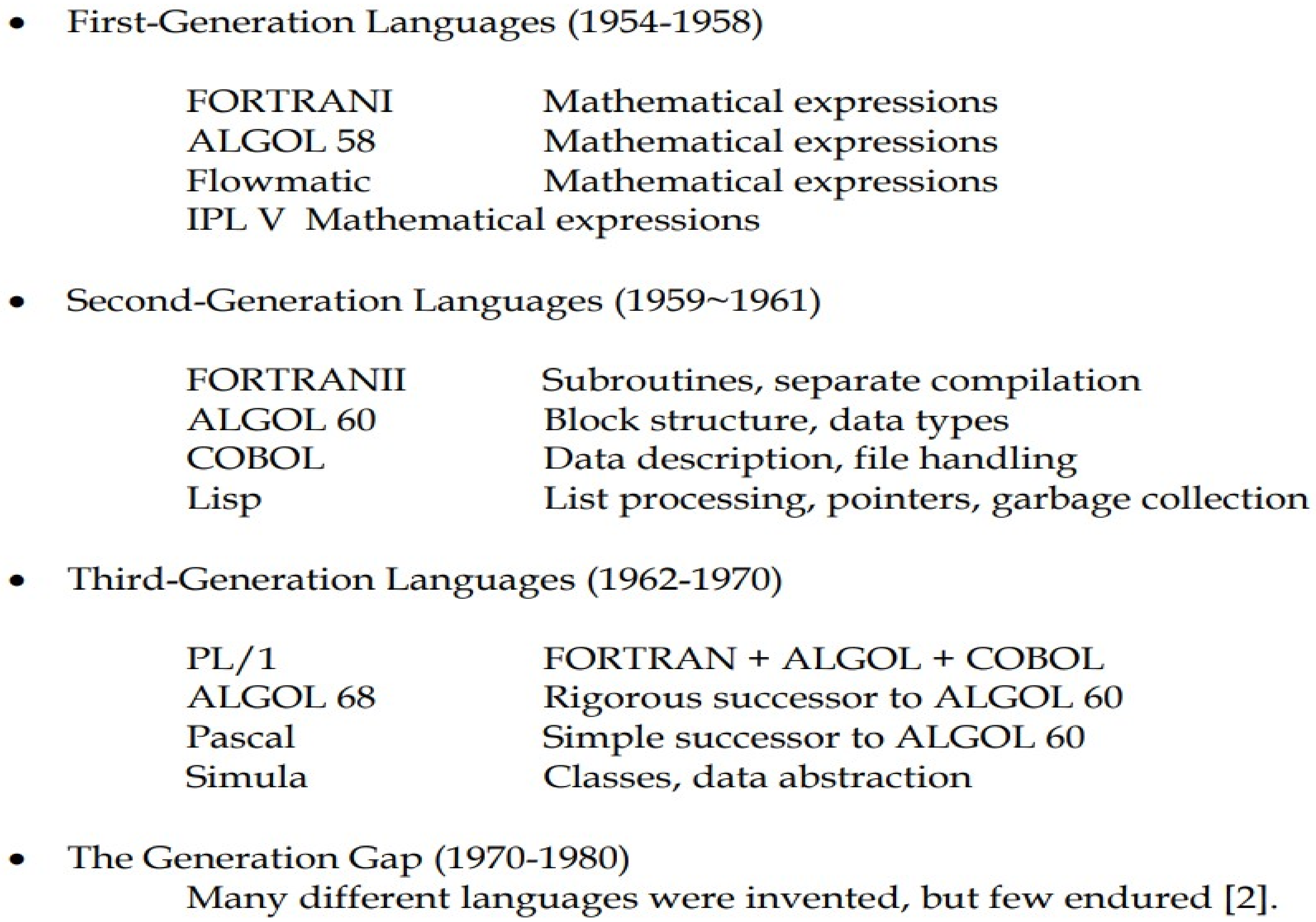
- It can be defined as a programming model which is derived from structured programming, based upon the concept of calling procedure.
- Procedures as (routines, subroutines or functions), simply consist of a series of computational steps to be carried out.
- During a program’s execution, any given procedure might be called at any point, including by other procedures or itself.

Procedural Oriented Programming	Object-Oriented Programming
The program is divided into small parts called **functions**.	The program is divided into small parts called **objects**.
Top-down approach	Bottom-up approach
Adding new data and functions is not easy	Adding new data and function is easy
Less secure	More secure
There is no concept of data hiding and inheritance	The concept of data hiding and inheritance is used.
The function is more important than the data.	Data is more important than function
Examples: C, FORTRAN, Pascal, Basic, etc.	**Examples:** C++, Java, Python, C#, etc.

Object Oriented Abstraction

- In Object-oriented programming, abstraction is a process of hiding the implementation details from the user, only the functionality will be provided to the user.
- It allows users not to get overwhelmed by the hidden logic that makes the users complex logic.
- Thus, OOPs abstracts hidden logic so that the user is not required to understand or even think about it.

The Evolution of the Object Model



- First-Generation Languages (1954-1958)

FORTRANI	Mathematical expressions
ALGOL 58	Mathematical expressions
Flowmatic	Mathematical expressions
IPL V	Mathematical expressions

- Second-Generation Languages (1959~1961)

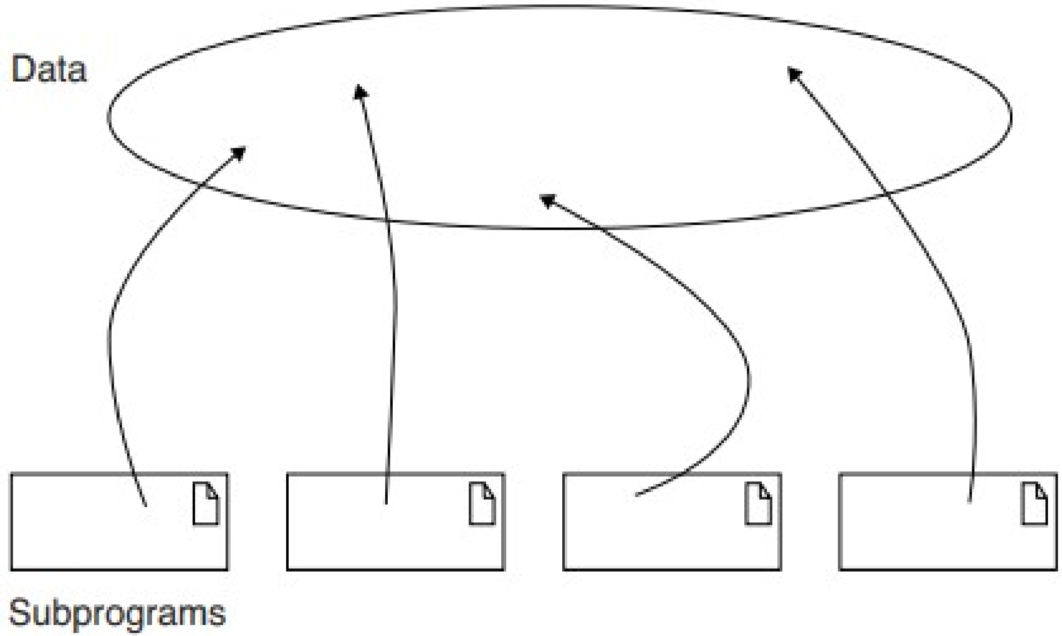
FORTRANII	Subroutines, separate compilation
ALGOL 60	Block structure, data types
COBOL	Data description, file handling
Lisp	List processing, pointers, garbage collection

- Third-Generation Languages (1962-1970)

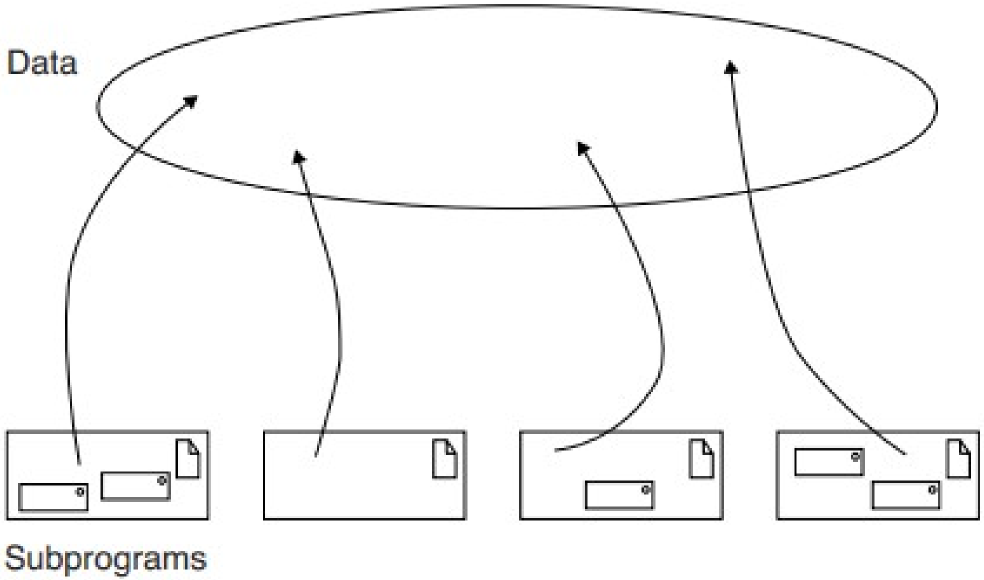
PL/1	FORTRAN + ALGOL + COBOL
ALGOL 68	Rigorous successor to ALGOL 60
Pascal	Simple successor to ALGOL 60
Simula	Classes, data abstraction

- The Generation Gap (1970-1980)
  Many different languages were invented, but few endured [2].

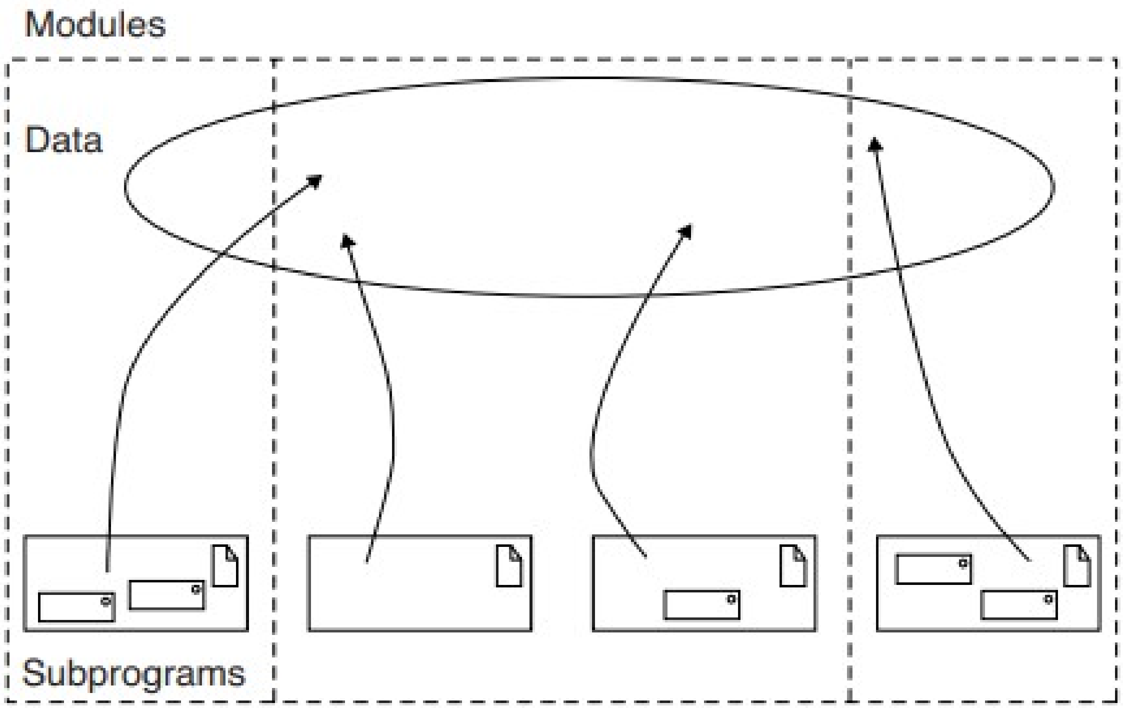
The Topology of First and Early Second-Generation Programming Languages


Data
Subprograms

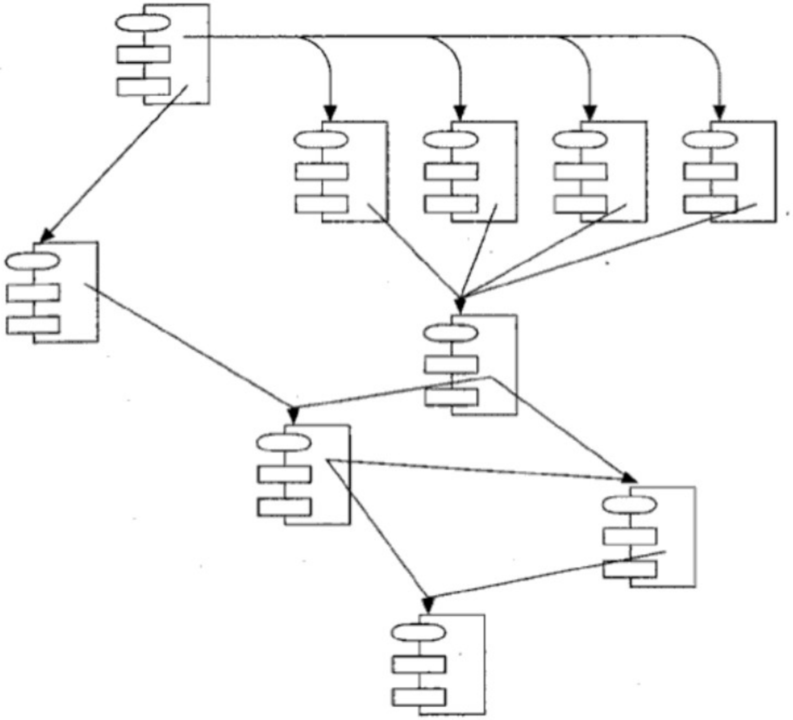
The Topology of Late Second and Early Third-Generation Programming Languages


Data
Subprograms

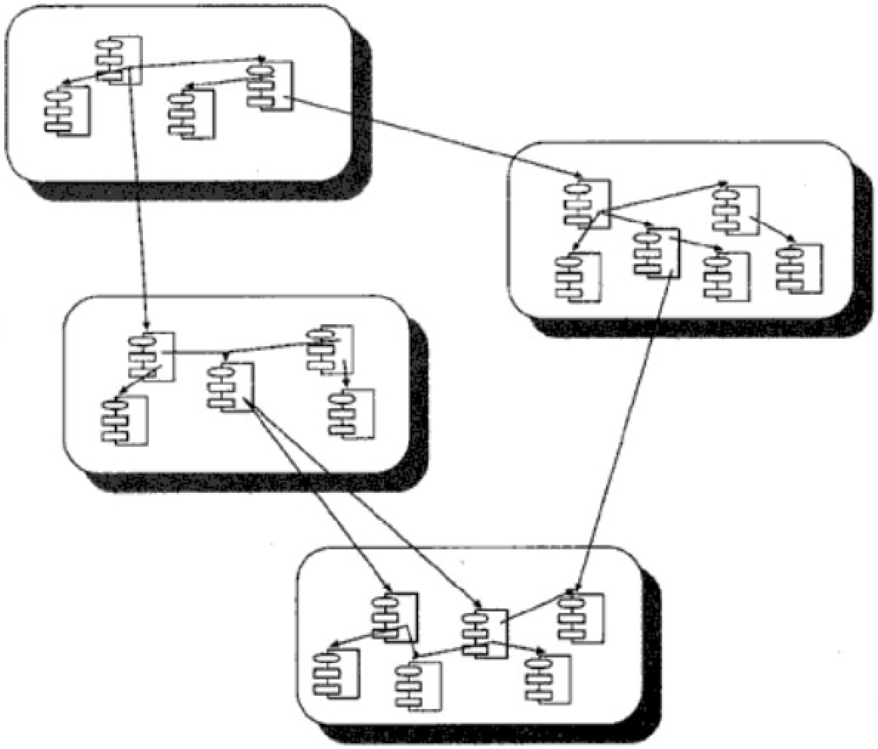
The Topology of Late Third-Generation Programming Languages


Modules
Data
Subprograms

The Topology of Small- to Moderate-Sized Applications using Object-Based and Object-Oriented Programming Languages



The Topology of Large Applications Using Object-Based and Object-Oriented Programming Languages



Foundations of the Object Model

- Object-oriented analysis and design represents an evolutionary development
- A **Smalltalk programmer** - methods
- A **C++ programmer** - virtual member functions
- A **CLOS programmer** - generic functions.
- An **Object Pascal programmer** -type coercion
- An **Ada programmer** calls the same thing a type conversion.
- ***Object-oriented Programming*** *– a method of implementation in which programs are organized as cooperative collections of objects, each of which represents an instance of some class, and whose classes are all members of a hierarchy of classes united via inheritance relationships.*

- A language is object-oriented if and only if it satisfies the following requirements:
  - It supports objects that are data abstractions with an interface of named operations and a hidden local state.
  - Objects have an associated type [class].
  - Types [classes] may inherit attributes from supertypes [superclasses]
- ***Object-oriented Design*** *- a method of design encompassing the process of object-oriented decomposition and a notation for depicting both logical and physical as well as static and dynamic models of the system under design*
- ***Object-oriented Analysis*** *- a method of analysis that examines requirements from the perspective of the classes and objects found in the vocabulary of the problem domain.*

Contributions to the Foundation of the Object Model

- "Advances in computer architecture, including capability systems and hardware support for operating systems concepts
- Advances in programming languages, as demonstrated in Simula, Smalltalk, CLU, and Ada
- Advances in programming methodology, including modularization and information hiding"
- Advances in database models
- Research in artificial intelligence
- Advances in philosophy and cognitive science

Elements of the Object Model

- There are five main kinds of programming styles:

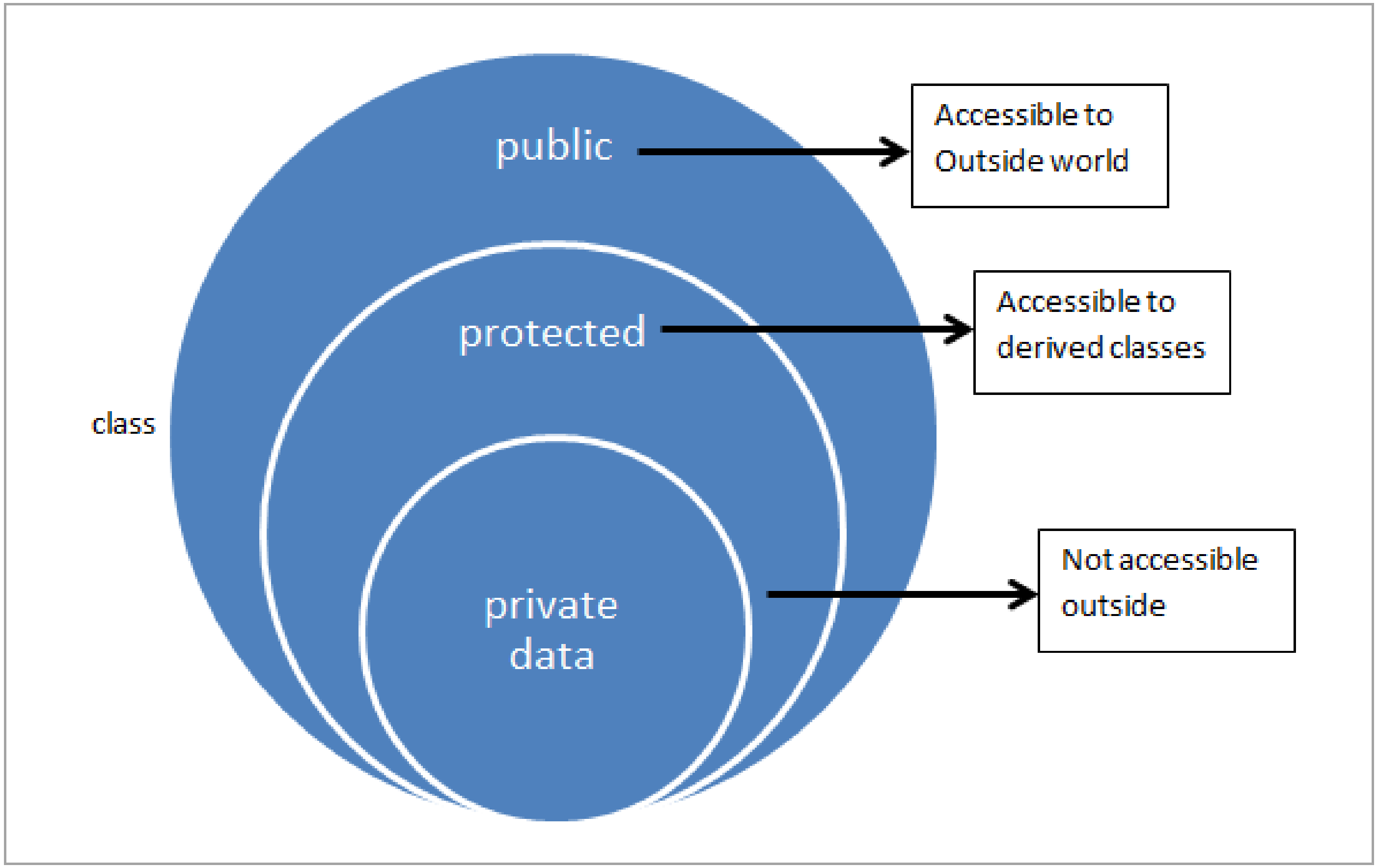
Procedure-oriented	Algorithms
Object-oriented	Classes and objects
Logic-oriented	Goals, often expressed in a predicate calculus
Rule-oriented	If-then rules
Constraint-oriented	Invariant relationships

- There is no single programming style that is best for all kinds of applications
- There are four major elements of this model: Abstraction, Encapsulation, Modularity & Hierarchy.

- ***Encapsulation*** *is the process of compartmentalizing the elements of an abstraction that constitute its structure and behavior; encapsulation serves to separate the contractual interface of an abstraction and its implementation.*
- ***Modularity*** *is the property of a system that has been decomposed into a set of cohesive and loosely coupled modules.*
- ***Hierarchy*** *is a ranking or ordering of abstractions.*
- ***Concurrency*** *is the properly that distinguishes an active object from one that is not active.*
- ***Persistence*** *is the property of an object through which its existence transcends time (i.e. the object continues to exist after its creator ceases to exist) and/or space (i. e. the objects location moves from the address space in which it was created).*

```
// c++ program to explain
// Encapsulation

#include<iostream>
using namespace std;

class Encapsulation
{
		private:
				// data hidden from outside world
				int x;

		public:
				// function to set value of
				// variable x
				void set(int a)
				{
						x =a;
				}

				// function to return value of
				// variable x
				int get()
				{
						return x;
				}
};

/
```

Encapsulation


public
protected
private
data
class

Accessible to Outside world

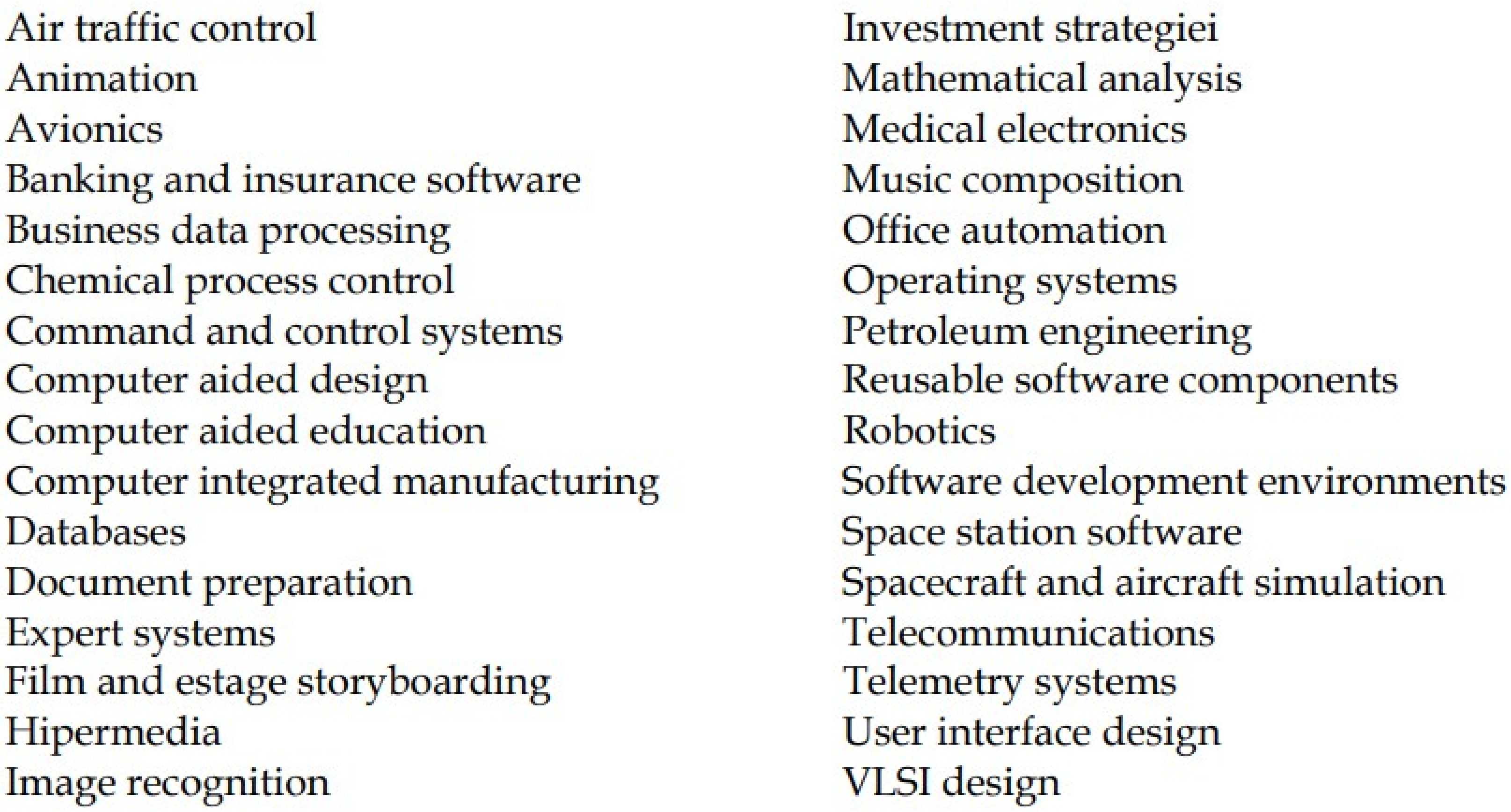
Accessible to derived classes

Not accessible outside

Applying the Object Model

- The use of the object model helps us to exploit the expressive power of object-based and object-oriented programming languages.
- Encourages the reuse not only of software but of entire designs, leading to the creation of reusable application frame-works
- Produces systems that are built upon stable intermediate forms, which are more resilient to change.

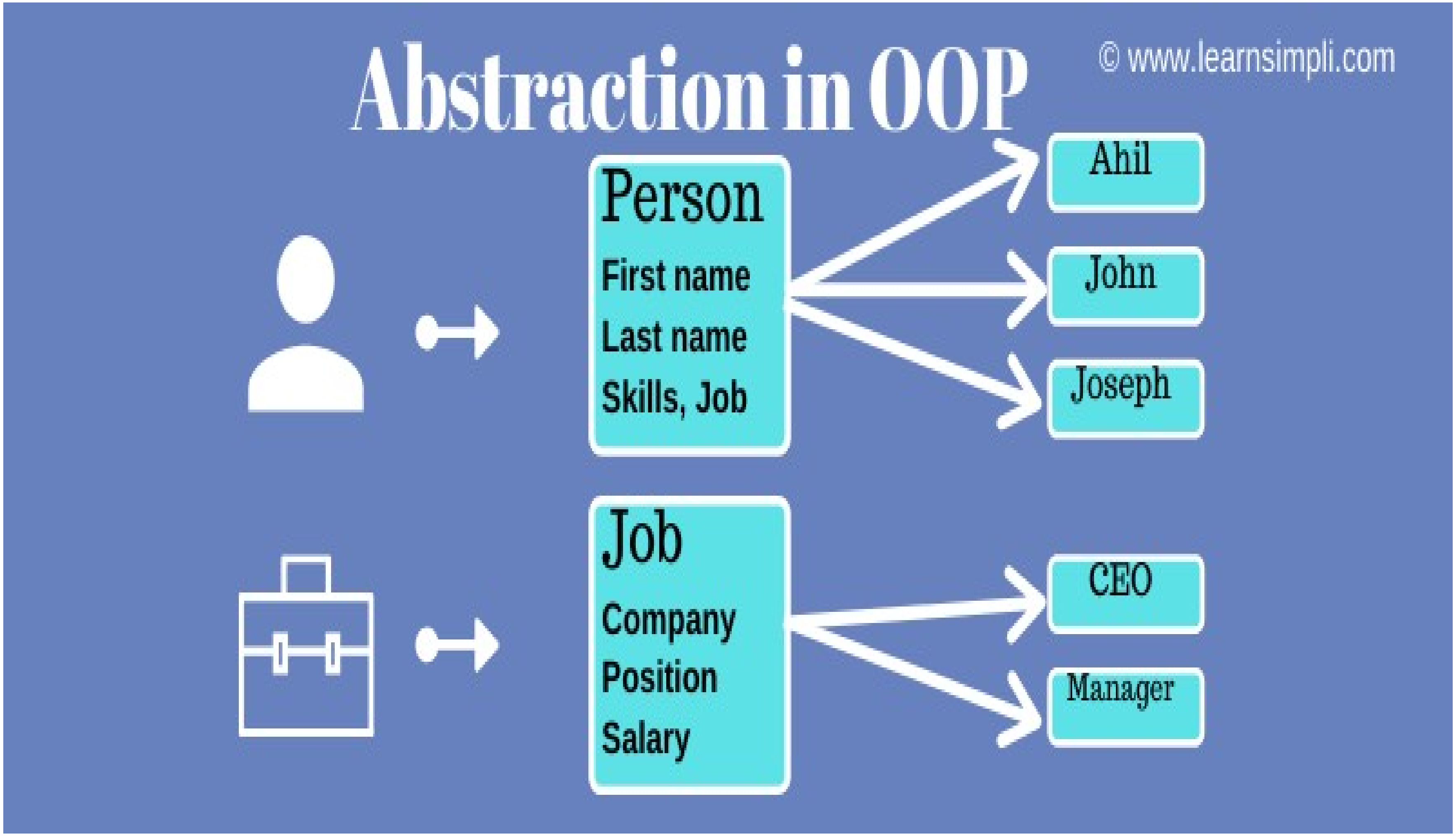
Applications of the Object Model



- Air traffic control
- Animation
- Avionics
- Banking and insurance software
- Business data processing
- Chemical process control
- Command and control systems
- Computer aided design
- Computer aided education
- Computer integrated manufacturing
- Databases
- Document preparation
- Expert systems
- Film and estage storyboarding
- Hipermedia
- Image recognition
- Investment strategiei
- Mathematical analysis
- Medical electronics
- Music composition
- Office automation
- Operating systems
- Petroleum engineering
- Reusable software components
- Robotics
- Software development environments
- Space station software
- Spacecraft and aircraft simulation
- Telecommunications
- Telemetry systems
- User interface design
- VLSI design

Abstractions in OOP

- Objects are the building blocks of Object-Oriented Programming.
- An object contains some properties and methods.
- It is possible to hide them from the outer world through access modifiers.
- Provide access only for required functions and properties to the other programs.
- This is the general procedure to implement abstraction in OOPS.
- **Abstract Class** is a type of class in OOPs, that declare one or more abstract methods.
- **Abstract Method** is a method that has just the method definition but does not contain implementation.

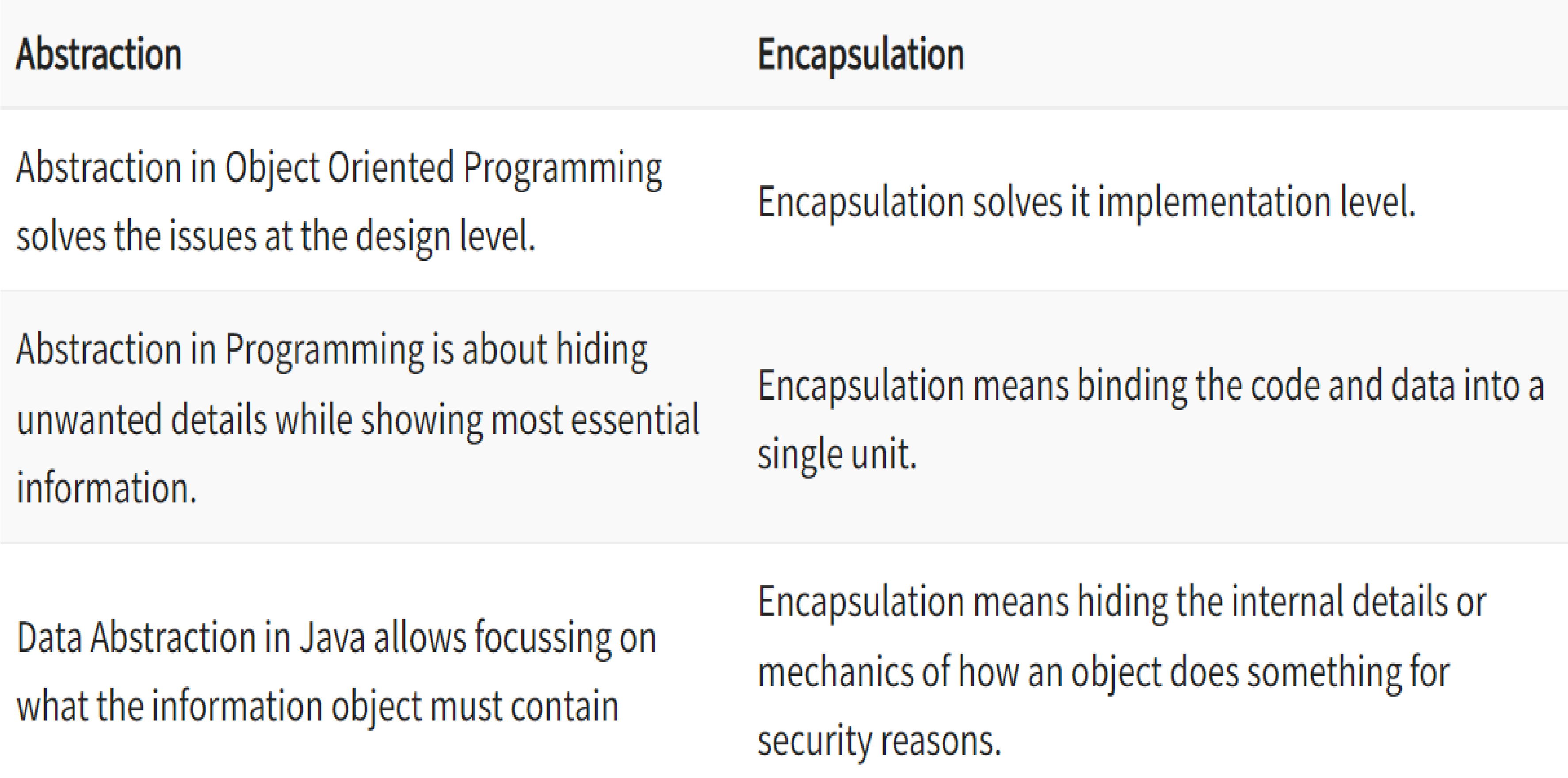

Abstraction in OOP
Person
First name
Last name
Skills, Job
Ahil
John
Joseph
Job
Company
Position
Salary
CEO
Manager

© www.learnsimpli.com

When to use Abstract Methods & Abstract Class?

- **Abstract methods** are mostly declared where two or more subclasses are also doing the same thing in different ways through different implementations.
- It extends the same Abstract class and offers different implementations of the abstract methods.
- **Abstract classes** help to describe generic types of behaviors and object-oriented programming class hierarchy.
- It describes subclasses to offer implementation details of the abstract class.

Difference between Abstraction and Encapsulation



Abstraction	Encapsulation
Abstraction in Object Oriented Programming solves the issues at the design level.	Encapsulation solves it implementation level.
Abstraction in Programming is about hiding unwanted details while showing most essential information.	Encapsulation means binding the code and data into a single unit.
Data Abstraction in Java allows focussing on what the information object must contain	Encapsulation means hiding the internal details or mechanics of how an object does something for security reasons.

Thank You

BITS Pilani
Pilani Campus

BITS Pilani
Pilani Campus

COMPUTING AND DESIGN

SESSION 14

Course design by - Dr. Lucy J. Gudino &
Dr Chandrasekhar
Faculty – Dr Thangakumar J
WILP & Department of CS & IS

BITS Pilani
Pilani Campus

Operating Systems – Software Design (Cont..)

Last Session

List of Topic Title	Text/Ref Book/external resource

• **The General Concept of Abstraction**
• **Types of Abstractions in Software Engineering**
  • Data Abstraction
  • Control Abstraction
• **Design Styles in Programming and Programming Languages: Procedural Design vs. OO Design**
  • Object Oriented Abstraction
  • The Evolution of the Object Model
  • Foundations of the Object Model
  • Elements of the Object Model
  • Applying the Object Model
  • Abstractions in OOP
	R4 (2.1, 2.2, 2.3, 2.4) and Class Notes

Software Design

Today's topic

- Inheritance and Delegation; Design with inheritance and delegation; Comparison: Program Composition – Object Hierarchy vs. Class Hierarchy
- OO Design and Design Patterns
- Design of Complex Software (Compilers/ OS/ Web Server as an example).

When to use Inheritance & Delegation?

Inheritance should only be used when:

- Both classes are in the same logical domain
- The subclass is a proper subtype of the superclass
- The superclass's implementation is necessary or appropriate for the subclass
- The enhancements made by the subclass are primarily additive.

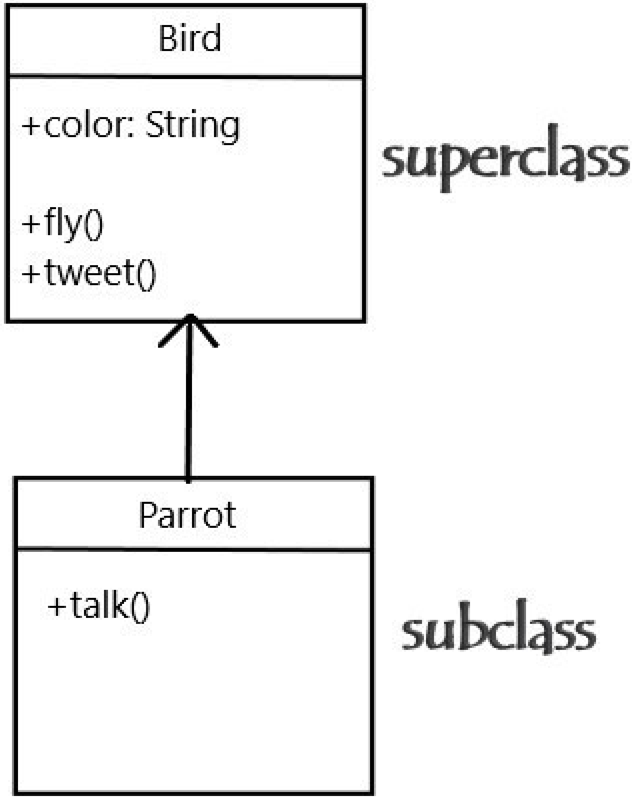
Delegation should only be used:

1. To represent or refer to one or more functions.
2. To define call-back methods.
3. To consume a delegate, we need to create an object to delegate.

Inheritance

- Inheritance defines a relationship among classes, wherein one class shares the structure or behavior defined in one or more classes.
- It represents a hierarchy of abstractions, in which a subclass inherits from one or more super classes.
- Typically, a subclass augments or redefines the existing structure and behavior of its super classes.
- It implies a generalization/ specialization hierarchy, wherein a subclass specializes the more general structure or behavior of its super classes.

Inheritance



Bird

+color: String

+fly()
+tweet()

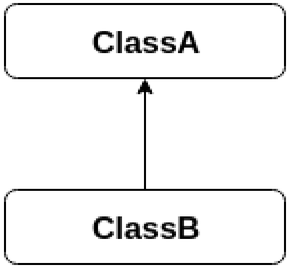
superclass

Parrot

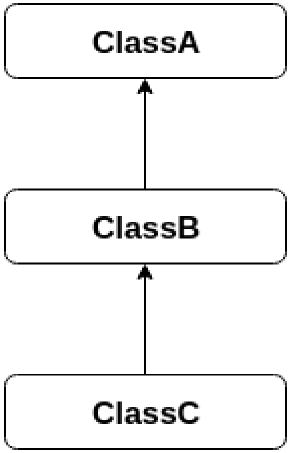
+talk()

subclass

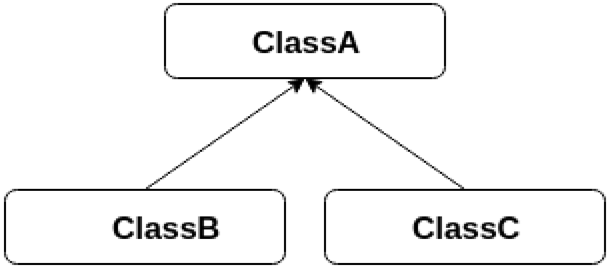
Types of Inheritance


ClassA
ClassB

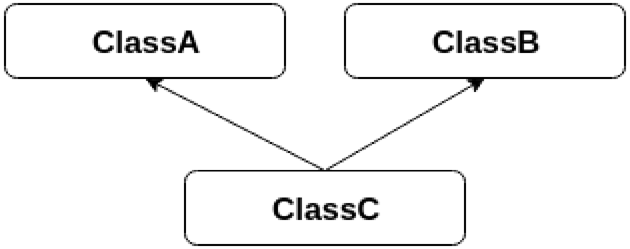
Single Inheritance


ClassA
ClassB
ClassC

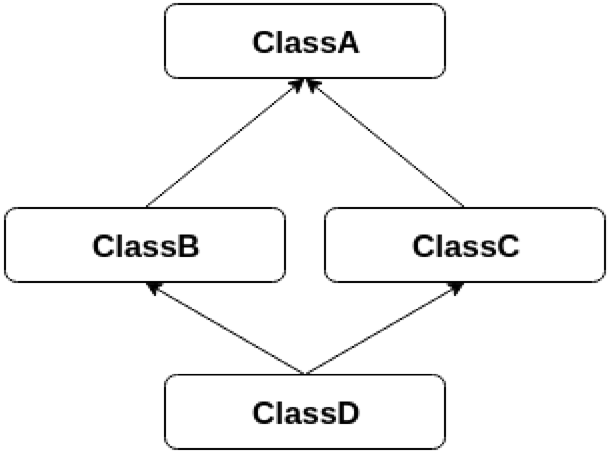
Multilevel Inheritance


ClassA
ClassB
ClassC

Hierarchical Inheritance

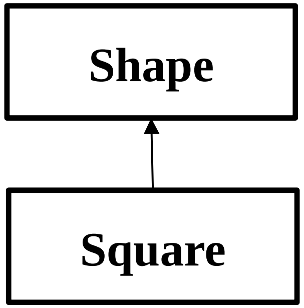

ClassA
ClassB
ClassC

Multiple Inheritance


ClassA
ClassB
ClassC
ClassD

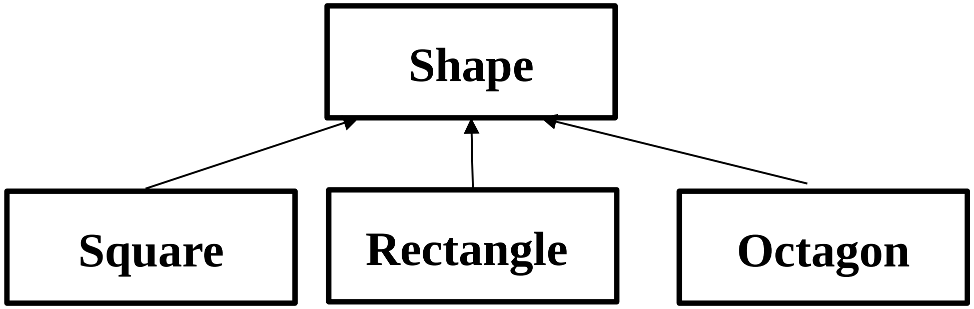
Hybrid Inheritance

Example


Shape

Square

Single Inheritance

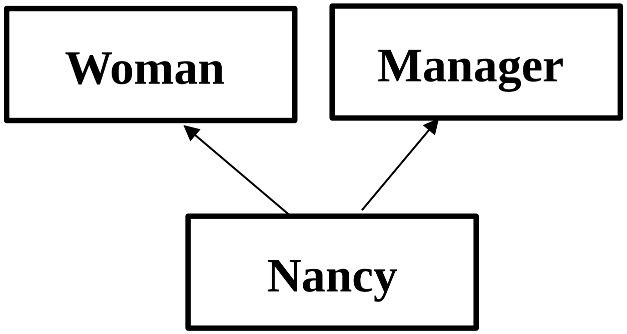

Shape

Square

Rectangle

Octagon

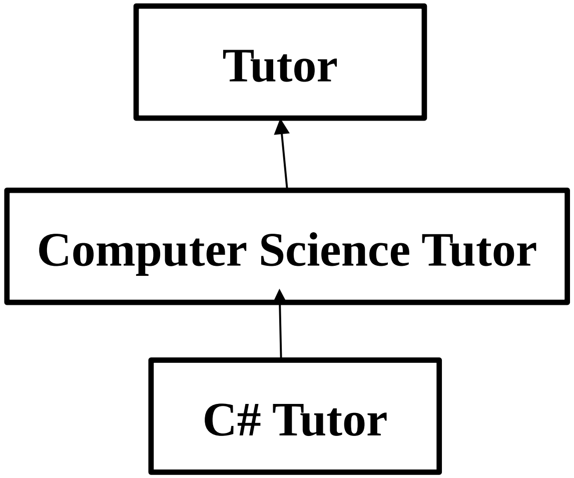
Hierarchical Inheritance


Woman

Manager

Nancy

Multiple Inheritance


Tutor

Computer Science Tutor

C# Tutor

Multi-Level Inheritance

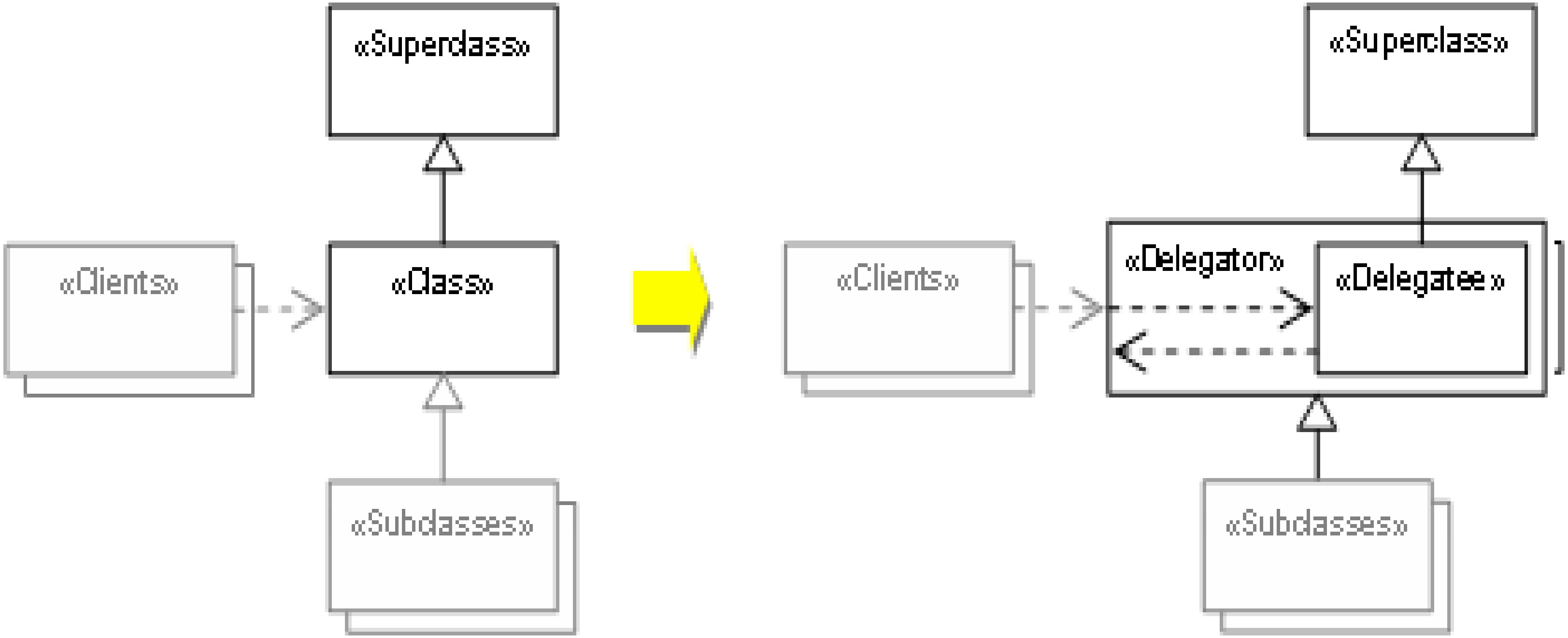
Use of Inheritance

- Used during object design
- Reduce redundancy
- Extensibility
- Specialization or generalization
- Used during requirements analysis

Delegation

- **Delegation** - objects delegate their behavior to related objects. i.e., The act of one object forwarding to another object an operation to be performed on behalf of the first object.
- **Advantage** - run-time flexibility
- **Disadvantage** - delegation is not directly supported by most popular object-oriented languages

Replacing Inheritance by Delegation

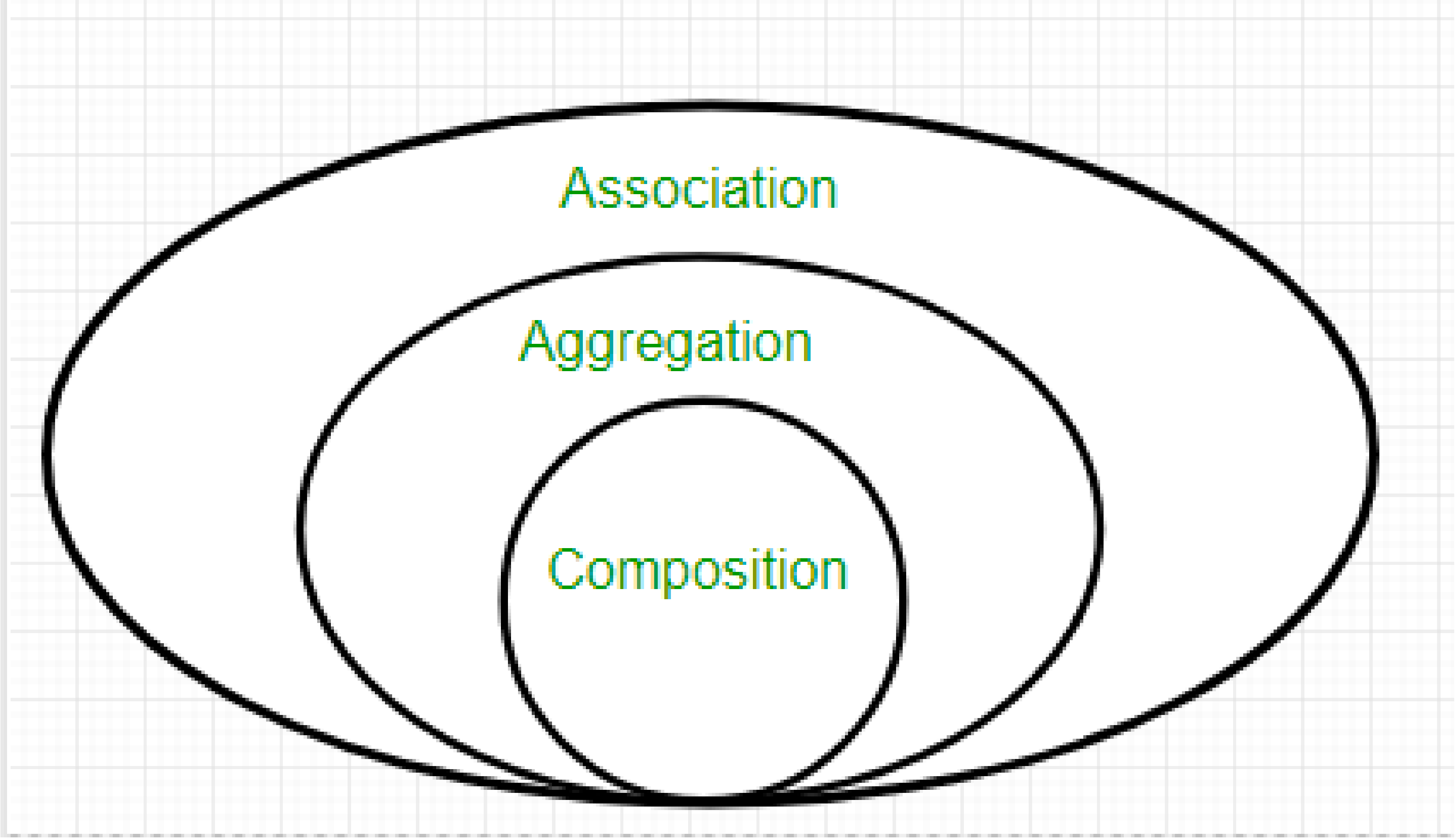

«Superclass»
«Clients»
«Class»
«Subclasses»
«Superclass»
«Clients»
«Delegator»
«Delegatee»
«Subclasses»

Delegation	Inheritance
Code sharing organization of objects	Code sharing organization of classes
Example: Java 1.1	Java 1.0
Adv: Flexibility, the reused class can be changed without changing reusing class	**Adv**: straightforward to use, Callbacks do not have to be public
Disadv: inefficiency	**Disadv**: exposes the subclass to the detail of its parent class

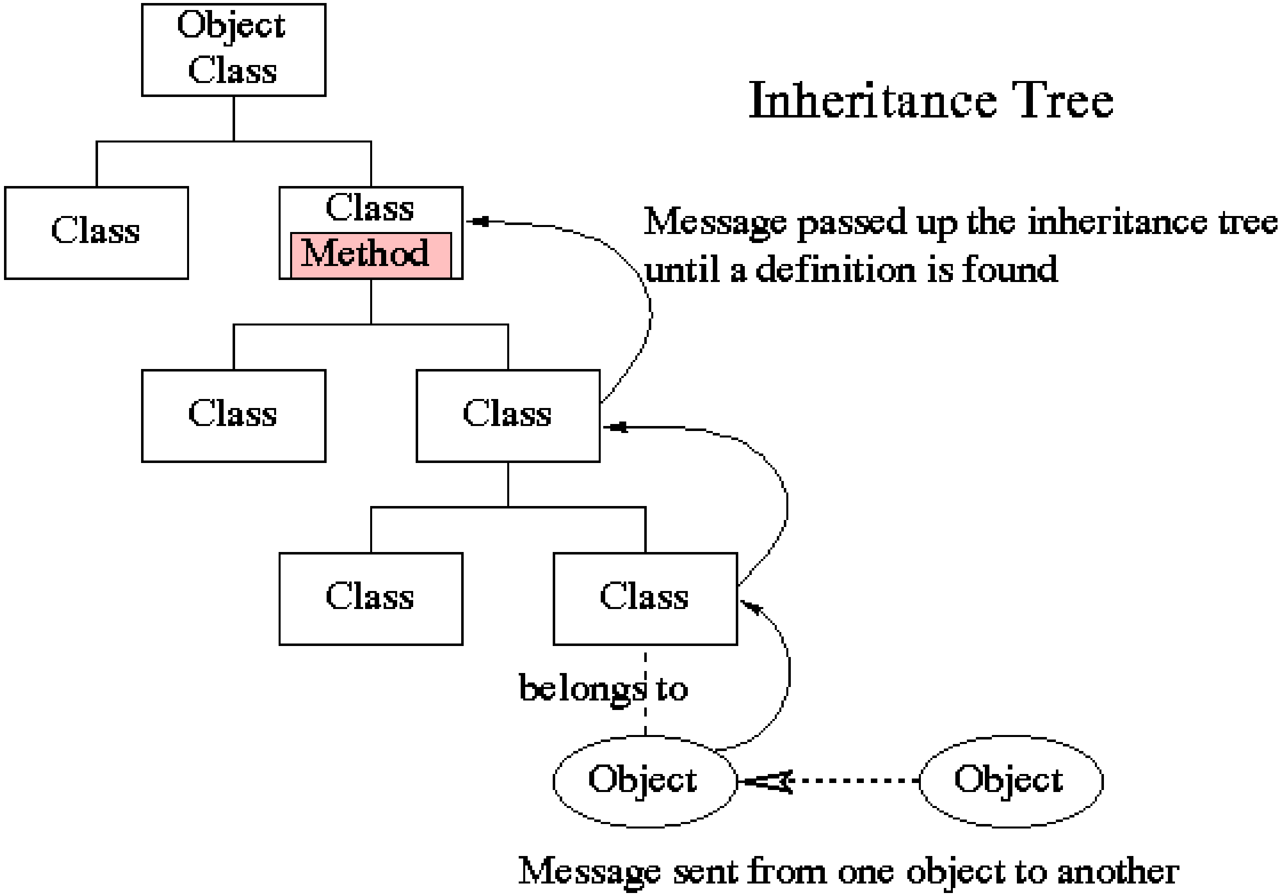
Program Composition

- **Composition** - describes a class that references one or more objects of other classes in instance variables. This allows to model a **has-a-relationship** association between objects.
- **Benefits -** reuse existing code, design clean APIs and change the implementation of a class used in a composition without adapting any external clients

Program Composition


Association
Aggregation
Composition

Class Hierarchy

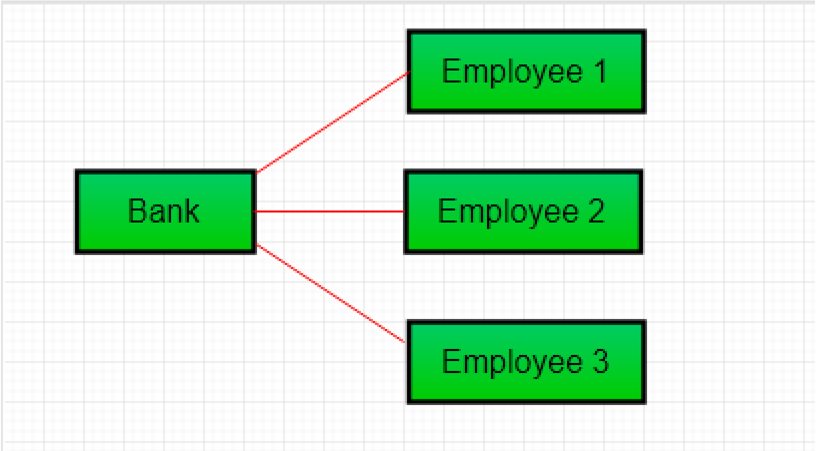

Object
Class
Class
Class
Method
Class
Class
Class
Class
belongs to
Object
Object

Inheritance Tree

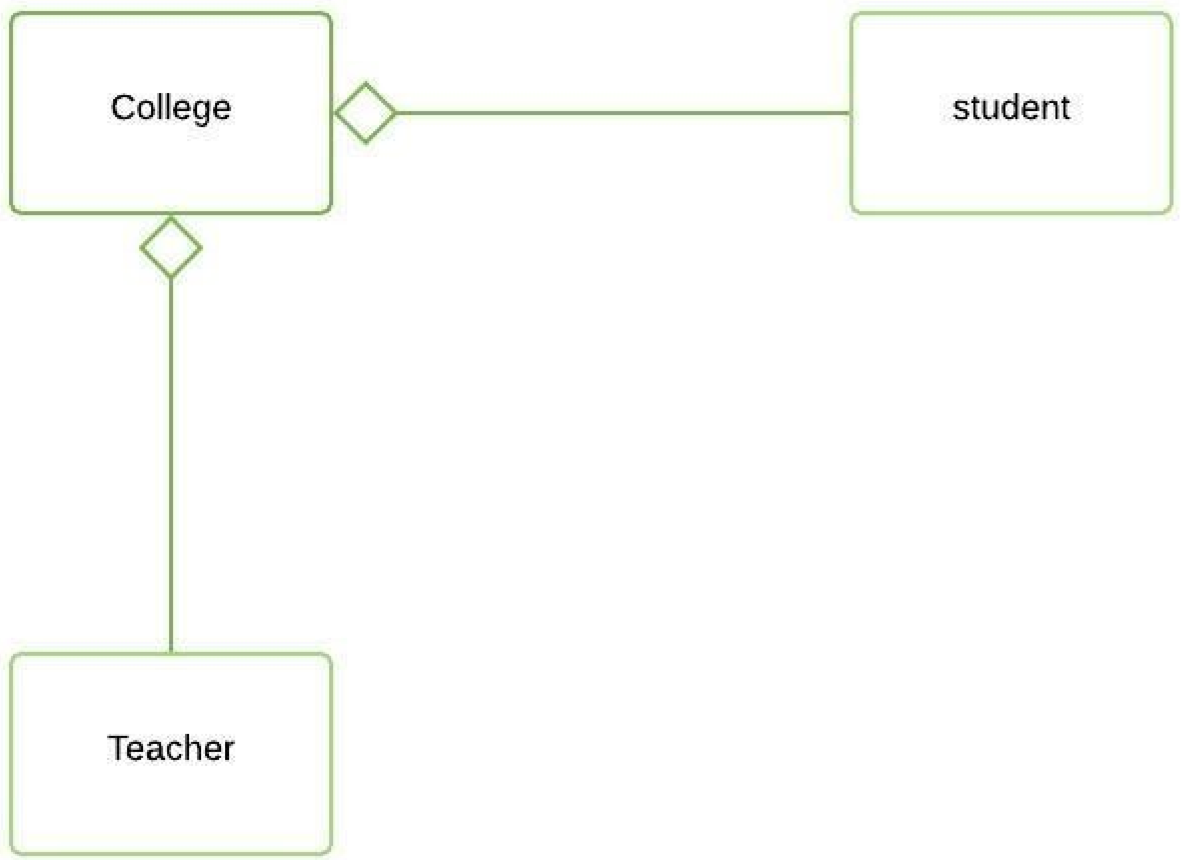
Message passed up the inheritance tree until a definition is found

Message sent from one object to another

Aggregation


Employee 1
Bank
Employee 2
Employee 3

Association


College
student
Teacher

Uses of Hierarchy

- An **object hierarchy** shows the order and the linkage objects possess when they link themselves
- A **class hierarchy** is an assembly of classes, a class hierarchy is popularly known as the hierarchy tree, it shows the relationship and order of objects.
- The depth of a **hierarchy** does not hinder the inheritance of a class variable and method down the hierarchical level

Object-oriented Design

- The users must be able to use objects of subclasses via references to base classes without noticing any difference.
- When using an object through its base class interface, the object of a subclass must not expect the user to obey preconditions that are stronger than those required by the base class.
- The five basic concepts of object-oriented design are the implementation level features that are built into the programming language: Object/Class, information hiding, inheritance, OOP and polymorphism.

Design Patterns

- Template solutions that developers have refined over time to solve a range of recurring problems.
- Elements of design patterns are:
  - **Name :** identifies pattern
  - **Problem description:** describes when to apply the pattern in terms of the problem and context
  - **Solution:** describes elements that make up the design, their relationship, responsibilities, and collaborations.
  - **Consequences:** results and trade-offs of applying the pattern

Gang of four in Design Patterns

- Design patterns are primarily based on the following principles of object orientated design.
  - Program to an interface not an implementation
  - Favor object composition over inheritance

Usage of Design Pattern

Design Patterns have two main usages in software development.

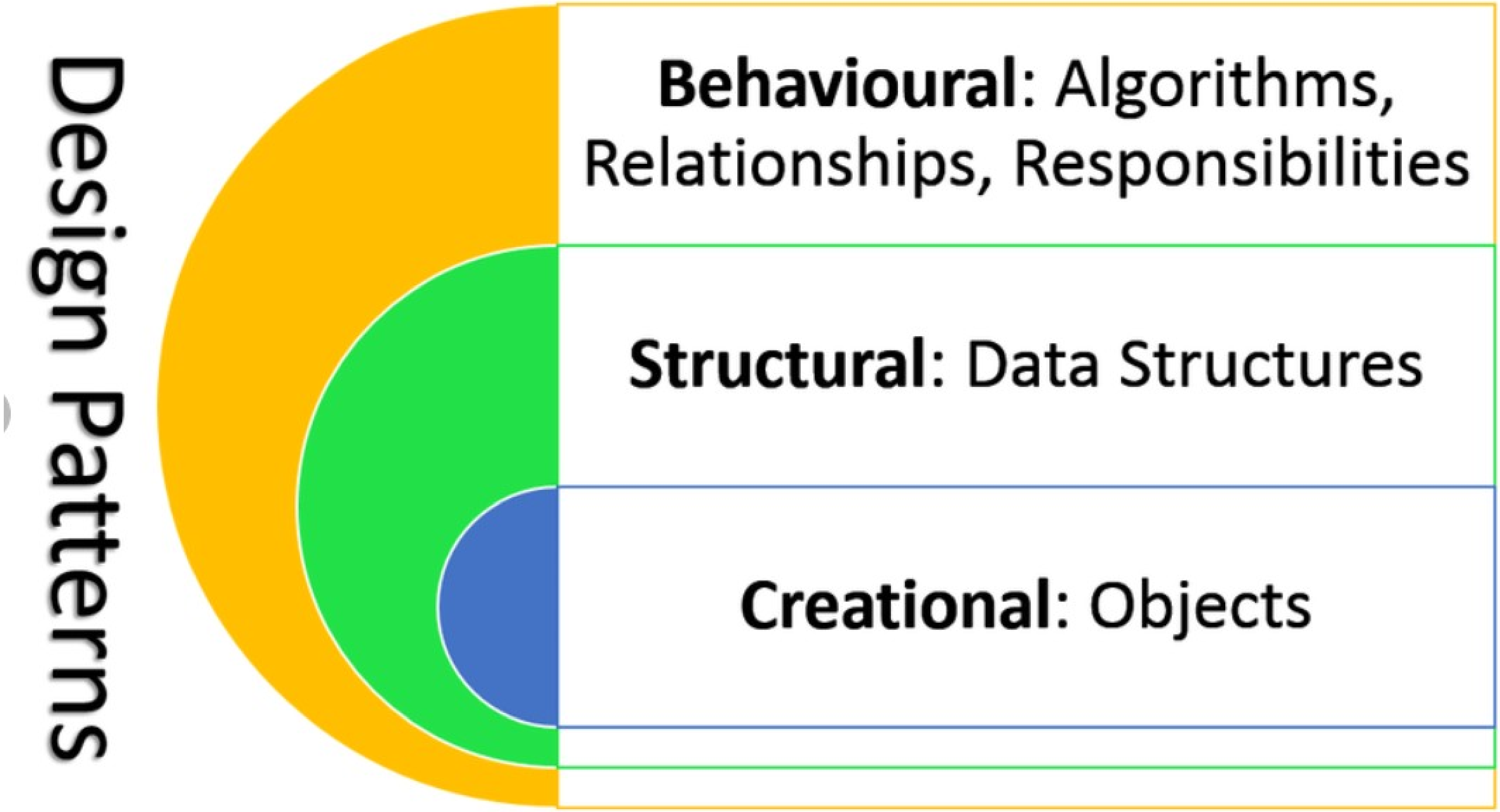
Common platform for developers

- Provide a standard terminology and are specific to particular scenario.

Best Practices

- Evolved over a long period of time and they provide best solutions to certain problems faced during software development.
- Learning these patterns helps un-experienced developers to learn software design in an easy and faster way.

Types of Design Patterns



Design Patterns

Behavioural: Algorithms, Relationships, Responsibilities

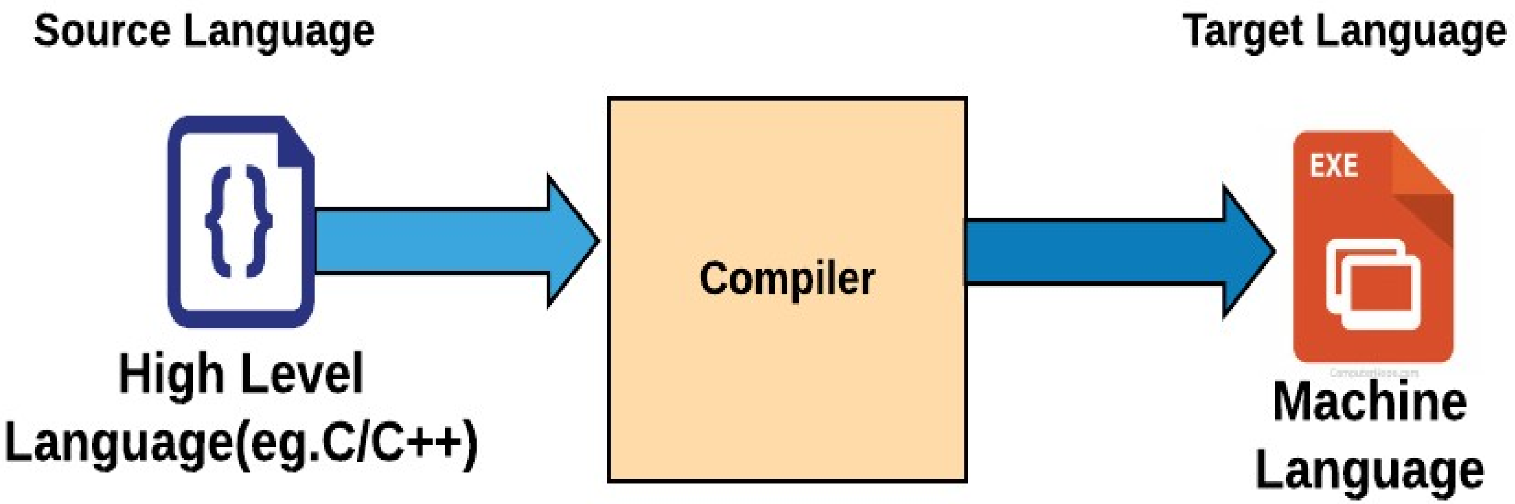
Structural: Data Structures

Creational: Objects

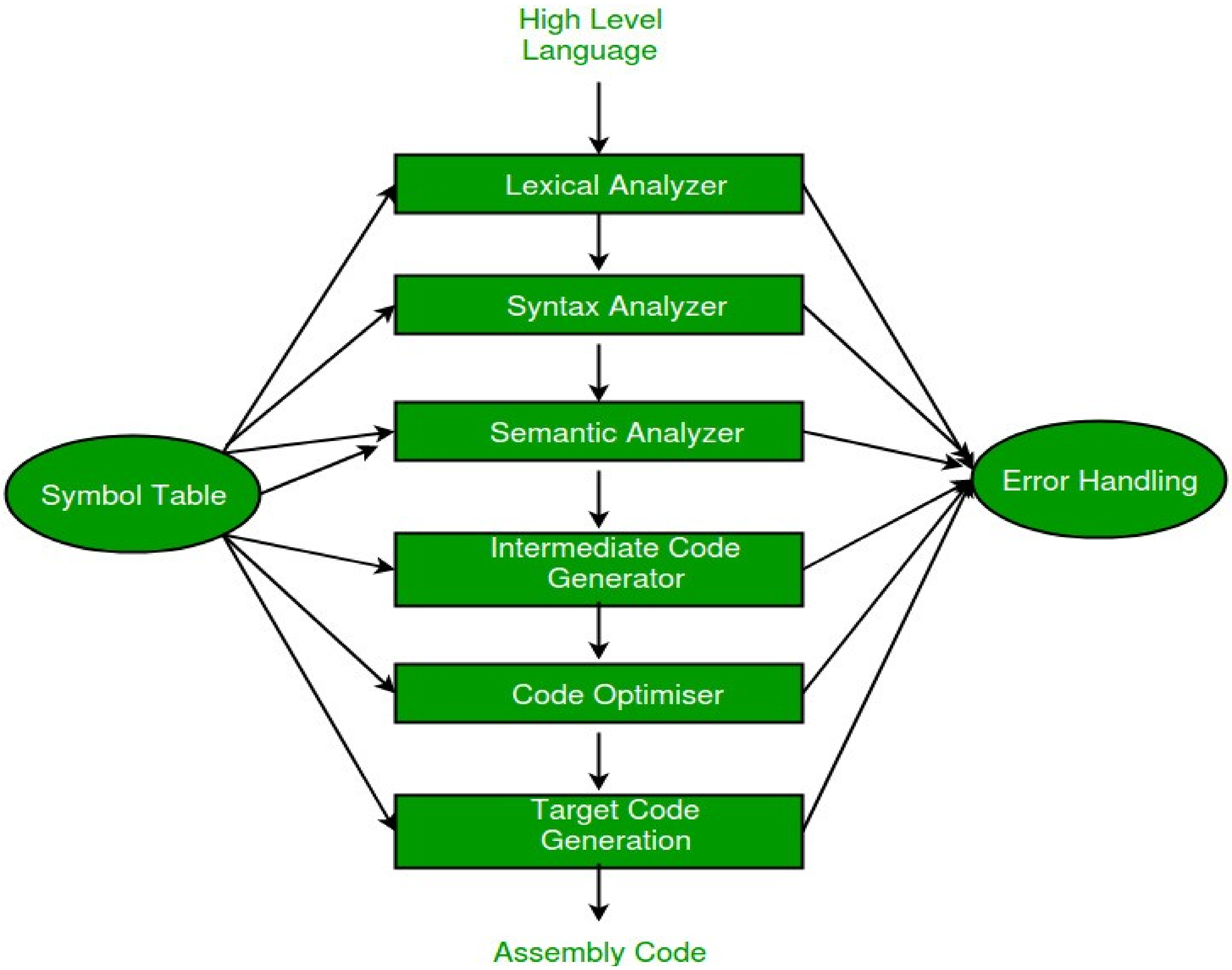
Advantages of Design Patterns

- More stable,
- Robust and
- Scalable
- Reusable in multiple projects
- Provide the solutions that help to define the system architecture
- Provide transparency to the design of an application

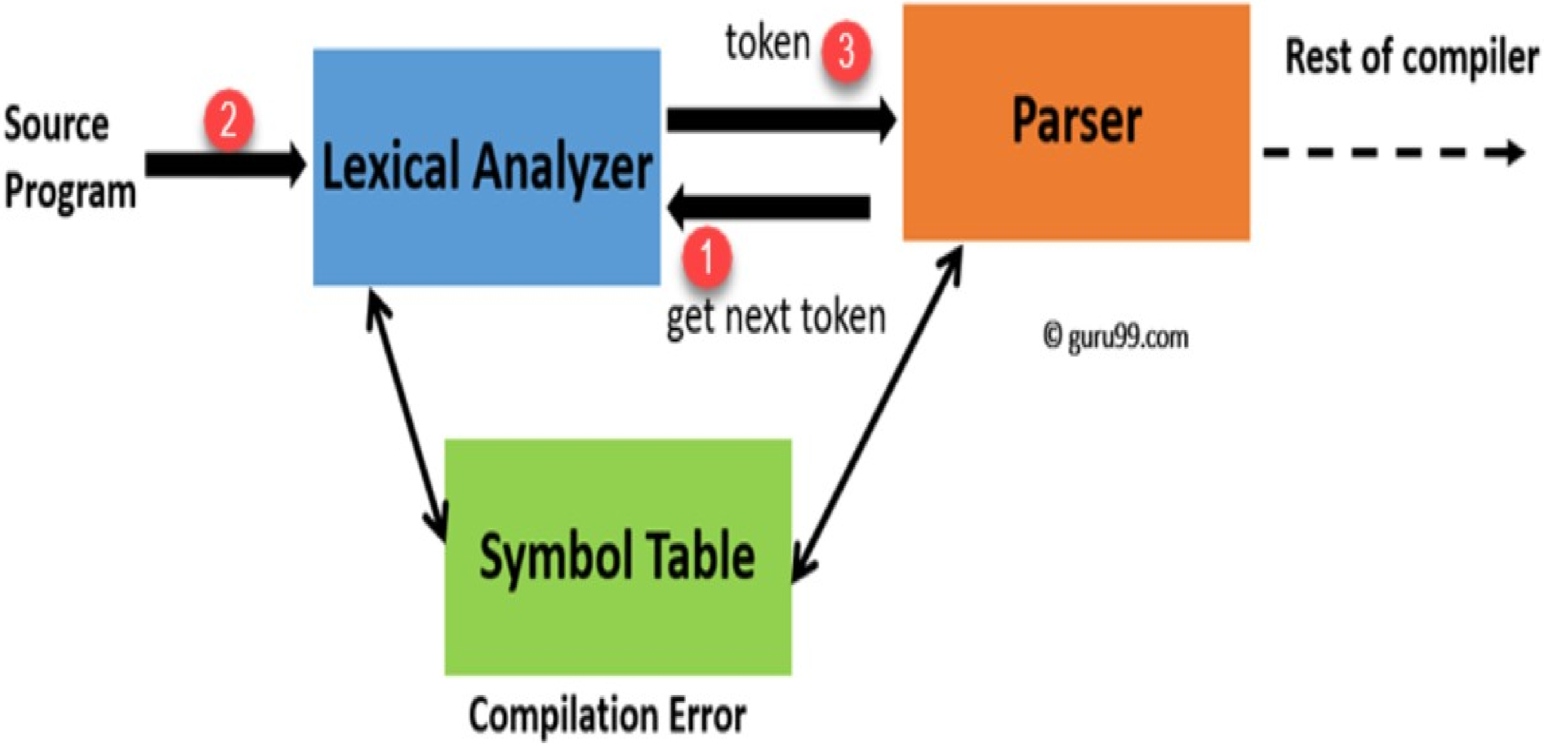
Design of Complex Software - Compiler


Source Language
Target Language
High Level Language(eg.C/C++)
Compiler
EXE
Machine Language

Phases of a Compiler


High Level Language
Lexical Analyzer
Syntax Analyzer
Semantic Analyzer
Intermediate Code Generator
Code Optimiser
Target Code Generation
Assembly Code
Symbol Table
Error Handling

Lexical Analyzer


Source Program
2
Lexical Analyzer
token
3
Parser
Rest of compiler
1
get next token
© guru99.com
Symbol Table
Compilation Error

Example of Lexical Analyzer in Compiler

- A statement through lexical analyzer –
- **a = b + c** ; It will generate token sequence like this:
- **id=id+id**; Where each **id** refers to it's variable in the symbol table referencing all details

```
int main()
{
// 2 variables
int a, b;
a = 10;
return 0;
}
```

- All the valid tokens are:

'int' 'main' '(' ')' '{' 'int' 'a' ',' 'b' ';' 'a' '=' '10' ';' 'return' '0' ';' '}'

Example of Syntax Analyzer in Compiler

- It checks the syntactical structure of the given input.
- The parse tree is constructed by using the pre-defined Grammar of the language and the input string.
- If the given input string can be produced with the help of the syntax tree (in the derivation process), the input string is found to be in the correct syntax. if not, error is reported by syntax analyzer.

S -> cAd
A -> bc|a
And the input string is “cad”

- Now the parser attempts to construct syntax tree from this grammar for the given input string. It uses the given production rules and applies those as needed to generate the string.

Example of Semantic Analyzer in Compiler

- It uses syntax tree and symbol table to check whether the given program is semantically consistent with language definition.
- It gathers type information and stores it in either syntax tree or symbol table.
- This type information is subsequently used by compiler during intermediate-code generation.

```
float x = 10.1;
float y = x*30;
```

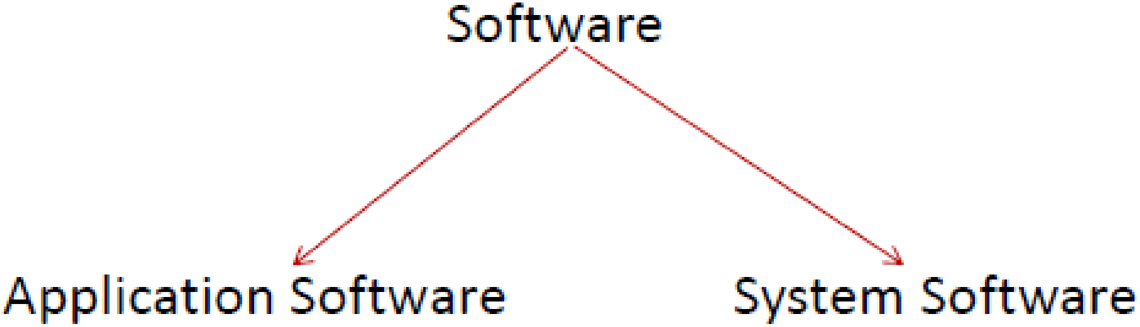
- In the above example integer 30 will be type casted to float 30.0 before multiplication, by semantic analyzer.

- Assemblers
- Assembler’s functions
- Assembler directives
- Object code structure
- Object code
- Two pass assembler
- Data structures for two pass assembler
- Operation code table
- Symbol table and location counter

Software

- software, is a collection of computer programs and related data that provide the instructions telling a computer *what to do* and *how to do it*

Classification



Software

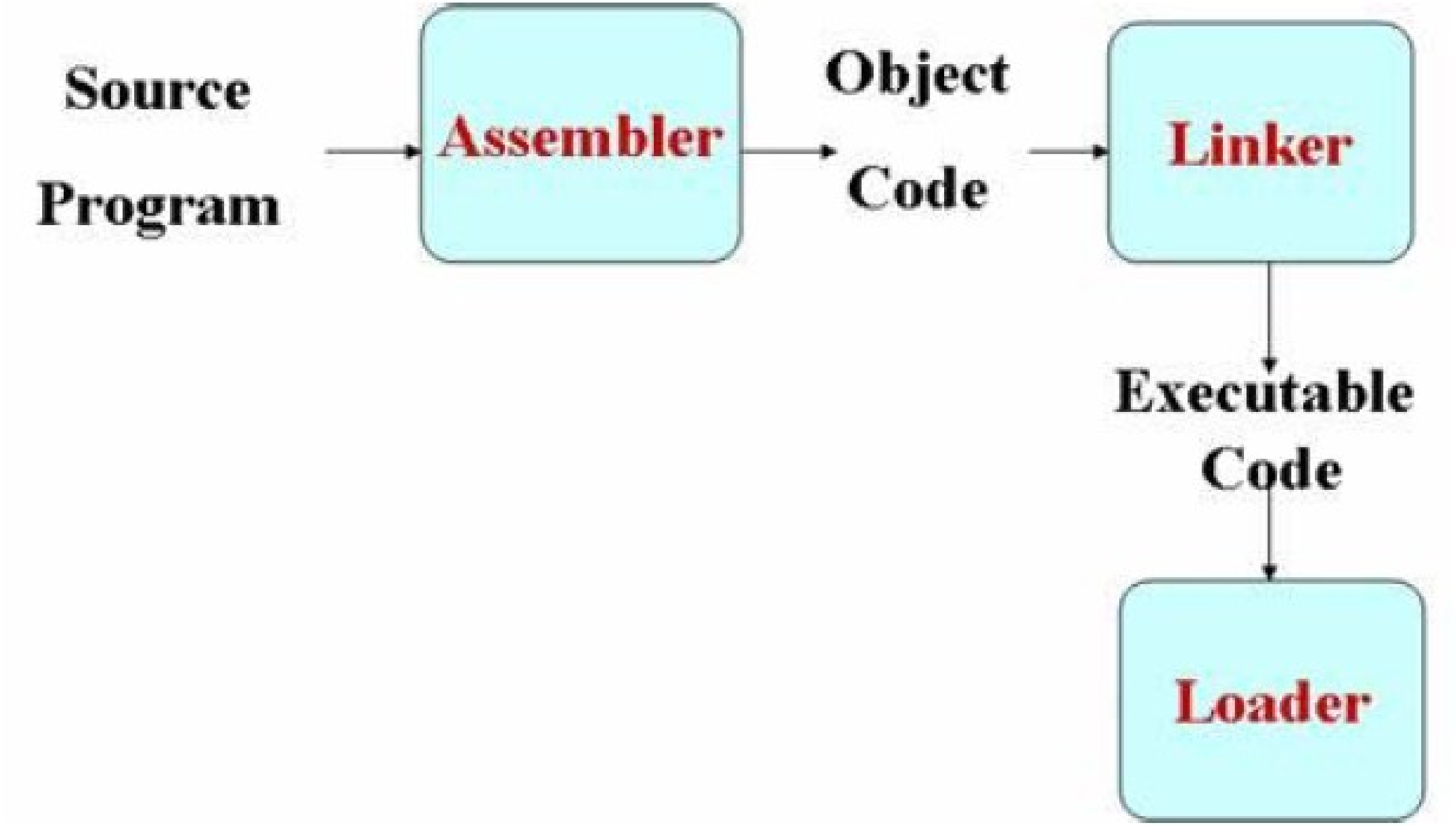
Application Software

System Software

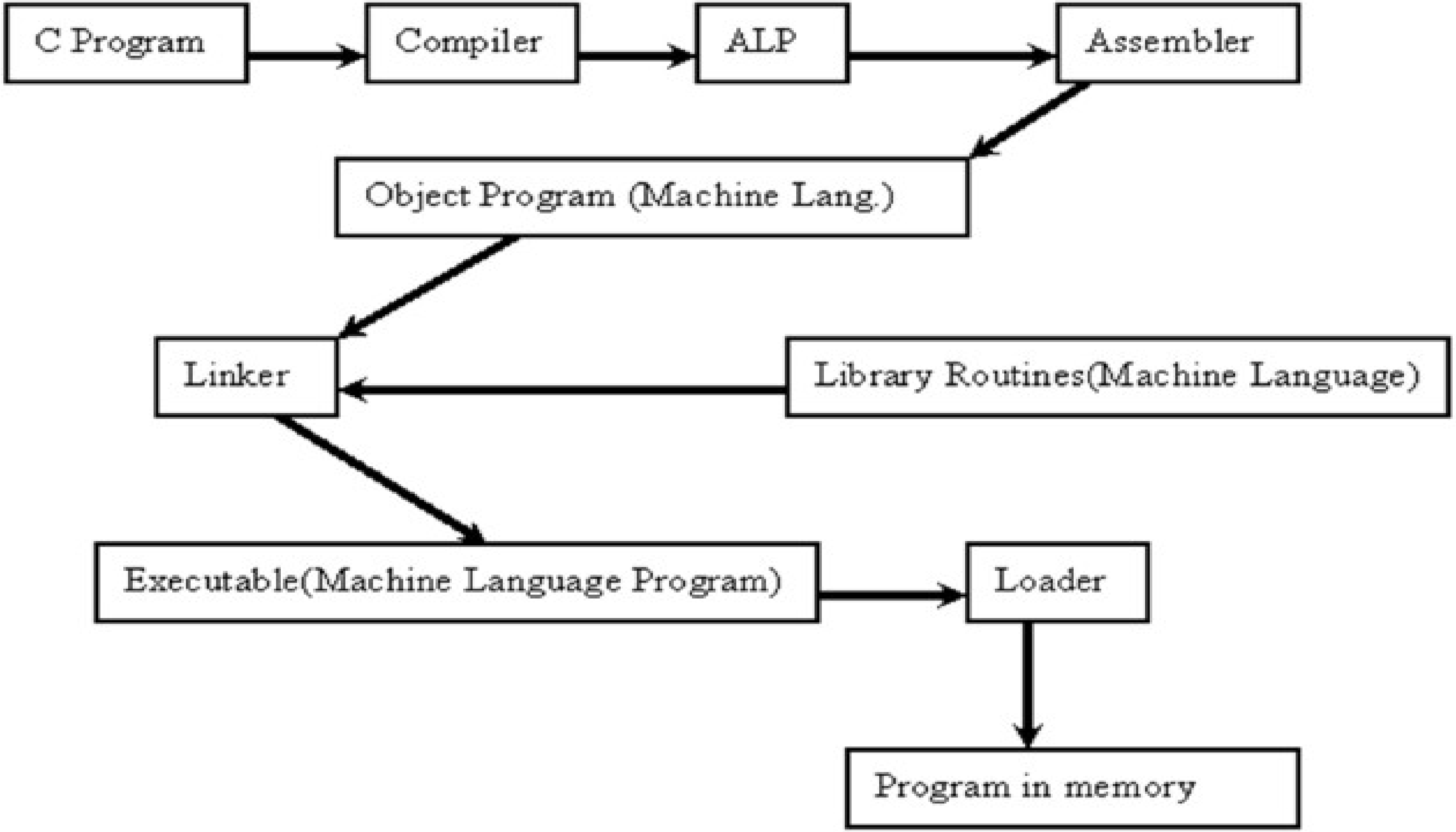
System software

- System software is computer software designed to operate the computer hardware and to provide a platform for running application software.
- Examples :
  - Loaders
  - Assembler
  - Linkers
  - Operating system
  - Compilers
  - Editors

Assemblers


Source
Program
Assembler
Object
Code
Linker
Executable
Code
Loader

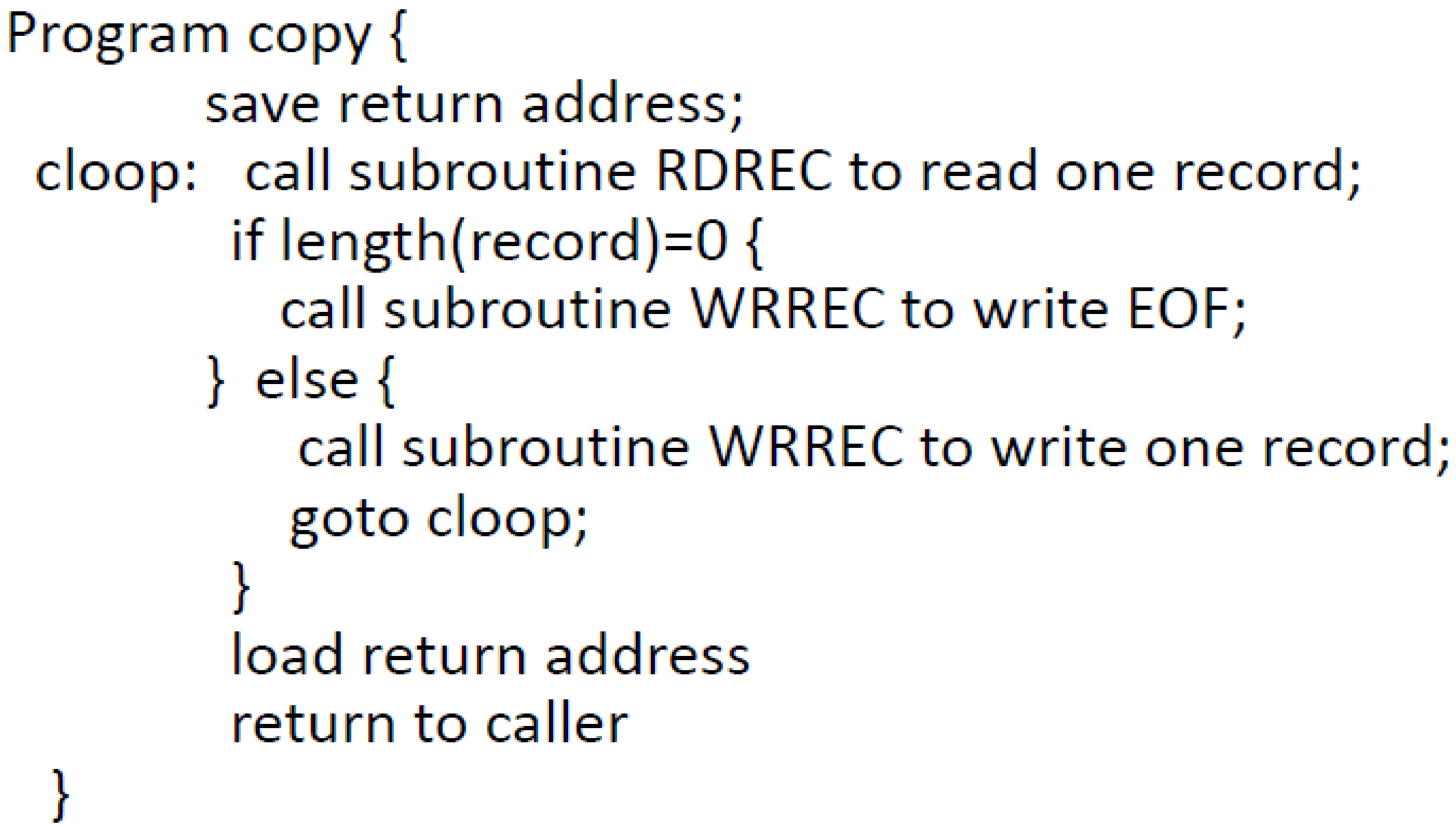
Assemblers


C Program
Compiler
ALP
Assembler
Object Program (Machine Lang.)
Linker
Library Routines(Machine Language)
Executable(Machine Language Program)
Loader
Program in memory

Assembler’s functions

- Convert mnemonic operation codes to their machine language equivalents
- Convert symbolic operands to their equivalent machine addresses
- Build the machine instructions in the proper format
- Convert the data constants to internal machine representations
- Write the object program and the assembly listing

Pseudo code



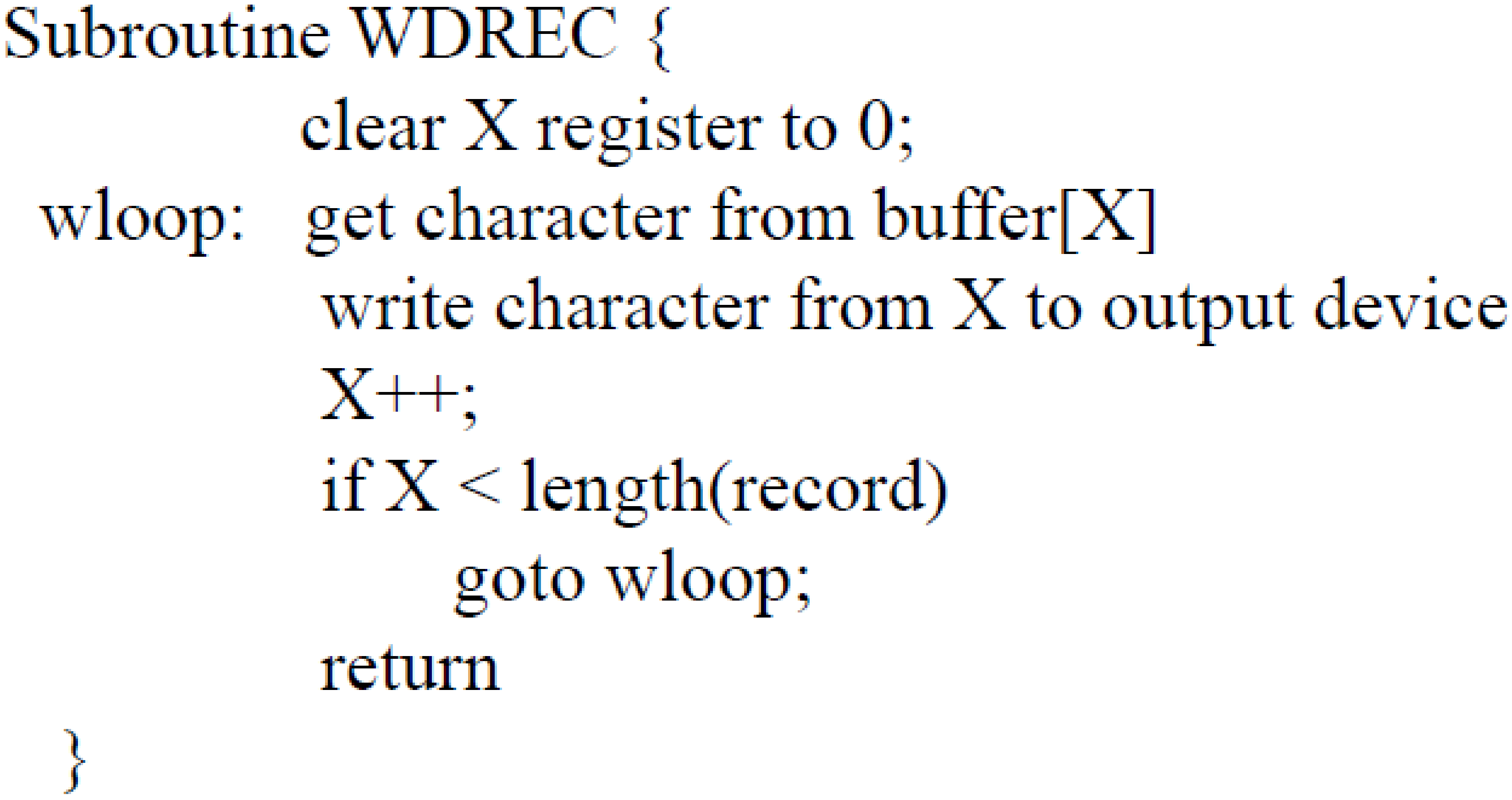
```
Program copy {
            save return address;
  cloop:   call subroutine RDREC to read one record;
             if length(record)=0 {
                call subroutine WRREC to write EOF;
            }  else {
                 call subroutine WRREC to write one record;
                goto cloop;
             }
             load return address
             return to caller
  }
```

Pseudo code

EOR:
character x‘00’

```
Subroutine RDREC {
        clear A, X register to 0;
 rloop:  read character from input device to A register
        if not EOR {
                store character into buffer[X];
                X++;
                if X < maximum length
                    goto rloop;
        }
        store X to length(record);
        return
}
```

Pseudo code

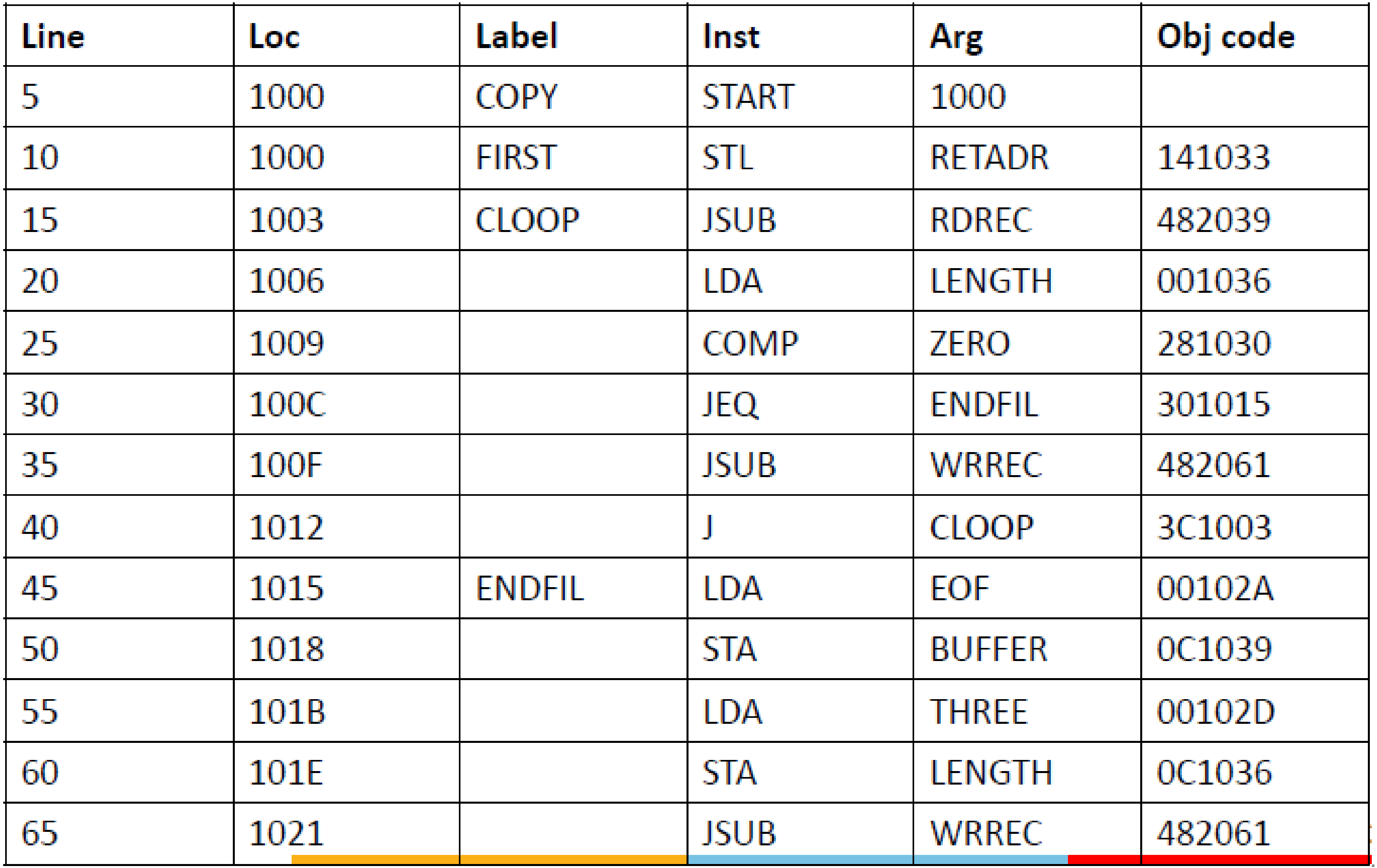


```
Subroutine WDREC {
            clear X register to 0;
  wloop:    get character from buffer[X]
             write character from X to output device
             X++;
             if X < length(record)
                  goto wloop;
             return
  }
```

Assembler Directives

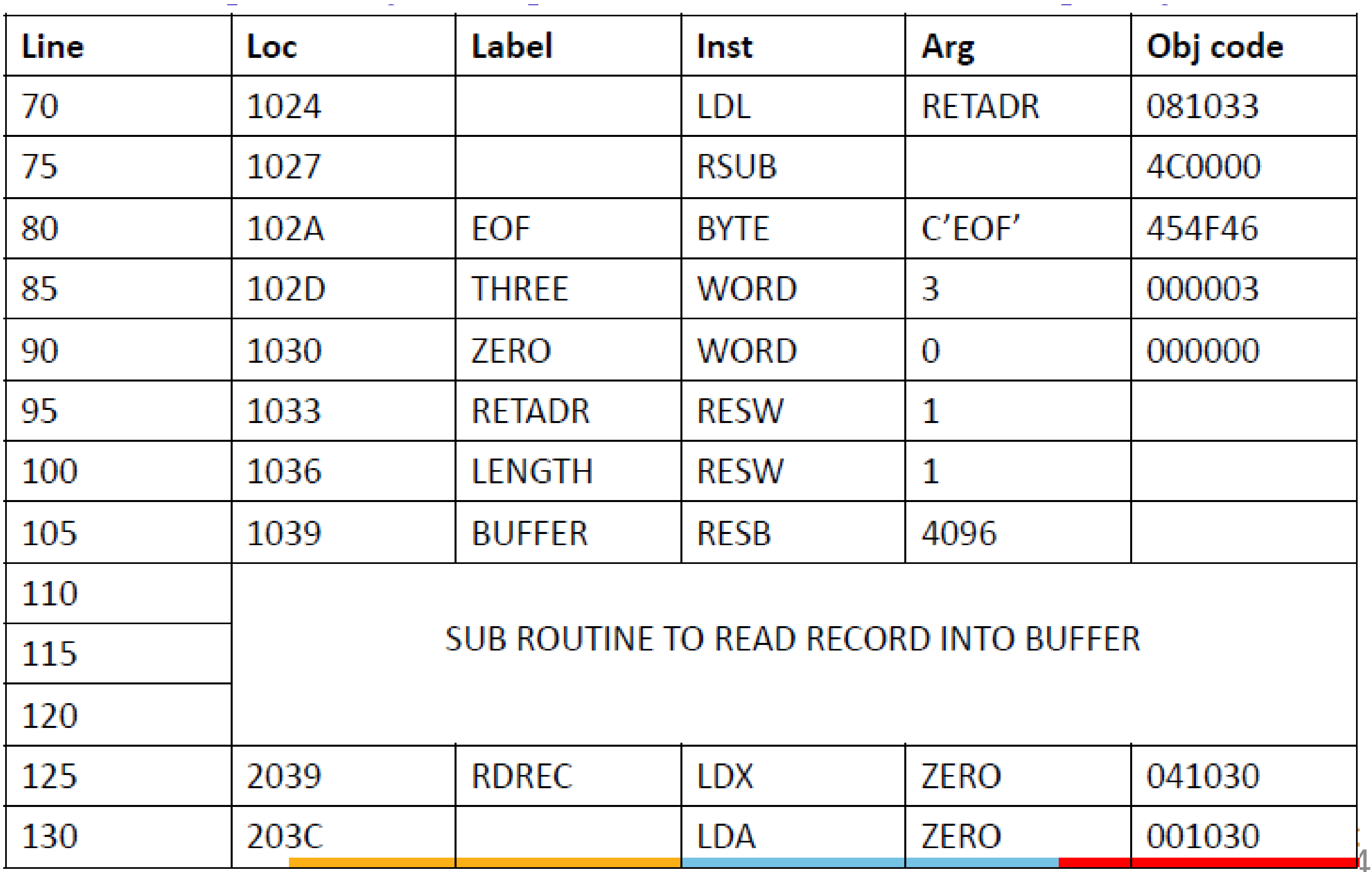
- Pseudo-Instructions
  - Not translated into machine instructions
  - Providing information to the assembler
- Basic assembler directives
  - START Specify name and starting address for the program
  - END end of the program
  - BYTE generate character or hexadecimal constant, occupying as many bytes as needed to represent the constant
  - WORD generate one-word integer constant
  - RESB reserve the indicated number of bytes for a data area
  - RESW reserve the indicated number of words for a data area

Program – takes input from input device



Line	Loc	Label	Inst	Arg	Obj code
5	1000	COPY	START	1000	
10	1000	FIRST	STL	RETADR	141033
15	1003	CLOOP	JSUB	RDREC	482039
20	1006		LDA	LENGTH	001036
25	1009		COMP	ZERO	281030
30	100C		JEQ	ENDFIL	301015
35	100F		JSUB	WRREC	482061
40	1012		J	CLOOP	3C1003
45	1015	ENDFIL	LDA	EOF	00102A
50	1018		STA	BUFFER	0C1039
55	101B		LDA	THREE	00102D
60	101E		STA	LENGTH	0C1036
65	1021		JSUB	WRREC	482061

Program – takes input from input device



Line	Loc	Label	Inst	Arg	Obj code
70	1024		LDL	RETADR	081033
75	1027		RSUB		4C0000
80	102A	EOF	BYTE	C'EOF'	454F46
85	102D	THREE	WORD	3	000003
90	1030	ZERO	WORD	0	000000
95	1033	RETADR	RESW	1	
100	1036	LENGTH	RESW	1	
105	1039	BUFFER	RESB	4096	
110	SUB ROUTINE TO READ RECORD INTO BUFFER				
115					
120					
125	2039	RDREC	LDX	ZERO	041030
130	203C		LDA	ZERO	001030

Assembler

- Translation of source program to object code requires following function
  1. Convert mnemonics operation codes to their machine language equivalents
     - E.g: Translate STL to 14
  2. Convert symbolic operands to their equivalent machine address
     - E.g: Translate RETADR to 1033
  3. Build the machine instructions in the proper format
  4. Convert the data constants specified in the source program into their internal machine representations
     - Translate EOF to 454F46
  5. Write the object program and assembly listing

Assembler

- All the steps except step 2 can easily be accomplished by sequential processing of the source program, one line at a time
- Step 2 can not be achieved by sequential processing
  - Contains forward reference
  - Assembler has to make 2 passes to resolve this
    - Pass 1 for label definition and assign address
    - Pass 2 for translation

Assembler

- Assembler functionality
  - Translate source program to object code
  - Process statements called assembler directives
    - Assembler directives are not translated into machine instructions
    - Assembler directives provide instructions to the assembler itself
    - Example: BYTE, WORD, RESB, RESW, START, END

Object Program

Simple object program format contains 3 types of records

- Header
  - Starting address, program name and length
- Text
  - Translated instructions and data of the program together with an indication of the addresses where these are to be loaded
- End
  - Marks the end of the program and specifies the address in the program where the execution is to begin

Object Program

Header

Col. 1	H
Col. 2~7	Program name
Col. 8~13	Starting address (hex)
Col. 14-19	Length of object program in bytes (hex)

Text

Col.1	T
Col.2~7	Starting address in this record (hex)
Col. 8~9	Length of object code in this record in bytes (hex
Col. 10~69	Object code (69-10+1)/6=10 instructions

End

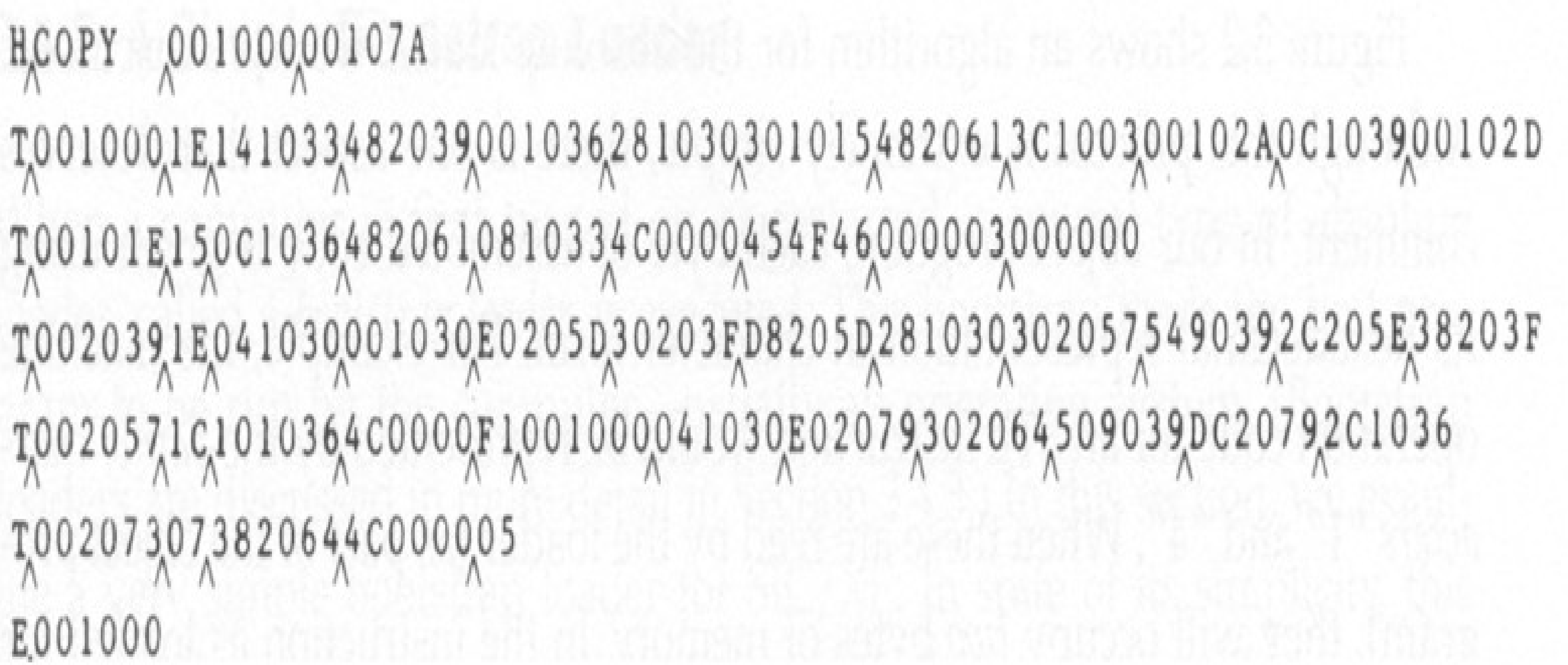
Col.1	E
Col.2~7	Address of first executable instruction (hex) (END program_name)

```
H COPY      001000 00107A

T 001000    1E     141033 482039 001036 281030 301015 482061 ...

T 00101E    15     0C1036 482061 081044 4C0000 454F46 000003 ...

T 002039    1E     041030 001030 E0205D 30203F D8205D 281030 ...

T 002057    1C     101036 4C0000 F1     001000 041030 E02079 ...

T 002073    07     382064 4C0000 05

E 001000
```

Object Code

```
H  COPY      001000  00107A

T  001000    1E      141033  482039  001036  281030  301015  482061 ...

T  00101E    15      0C1036  482061  081044  4C0000  454F46  000003 ...

T  002039    1E      041030  001030  E0205D  30203F  D8205D  281030 ...

T  002057    1C      101036  4C0000  F1      001000  041030  E02079 ...

T  002073    07      382064  4C0000  05

E  001000
```

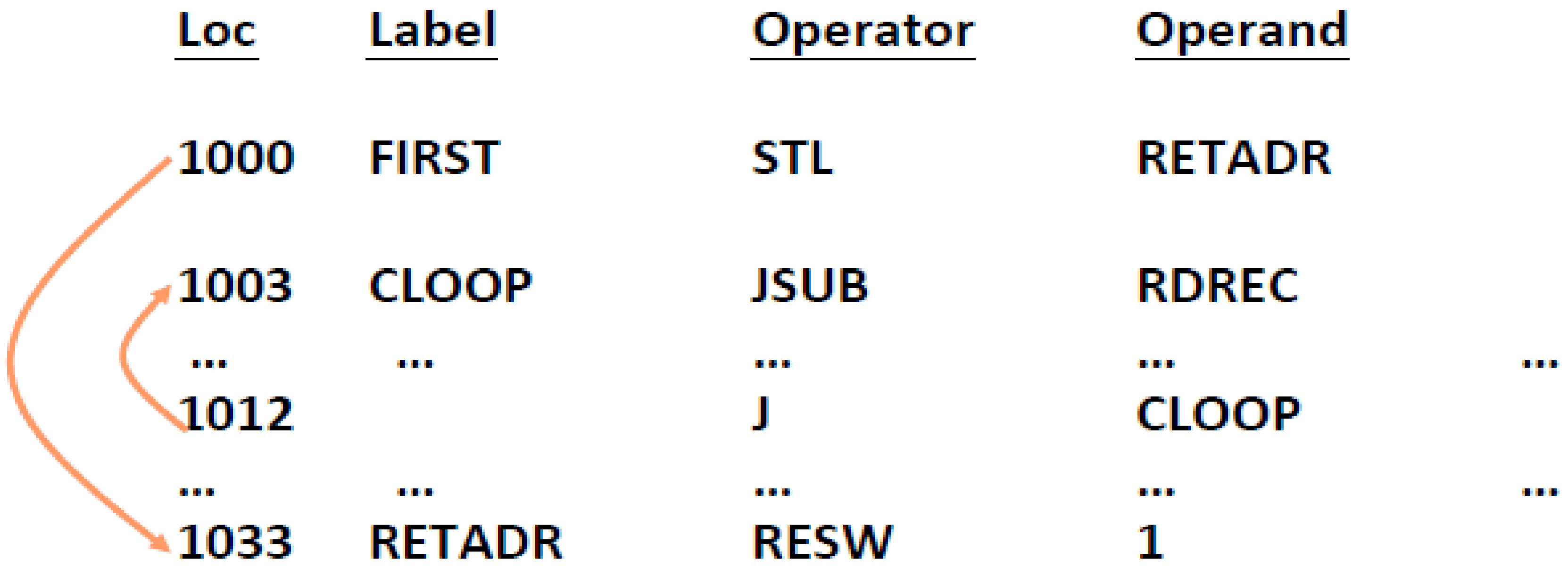
Object Code



```
H^COPY  ^001000^00107A
T^001000^1E^141033^482039^001036^281030^301015^482061^3C1003^00102A^0C1039^00102D
T^00101E^15^0C1036^482061^081033^4C0000^454F46^000003^000000
T^002039^1E^041030^001030^E0205D^30203F^D8205D^281030^302057^549039^2C205E^38203F
T^002057^1C^101036^4C0000^F1^001000^041030^E02079^302064^509039^DC2079^2C1036
T^002073^07^382064^4C0000^05
E^001000
```

Difficulties: Forward Reference

Forward reference: reference to a label that is defined later in the program.



Loc	Label	Operator	Operand	
1000	FIRST	STL	RETADR	
1003	CLOOP	JSUB	RDREC	
…	…	…	…	…
1012		J	CLOOP	
…	…	…	…	…
1033	RETADR	RESW	1	

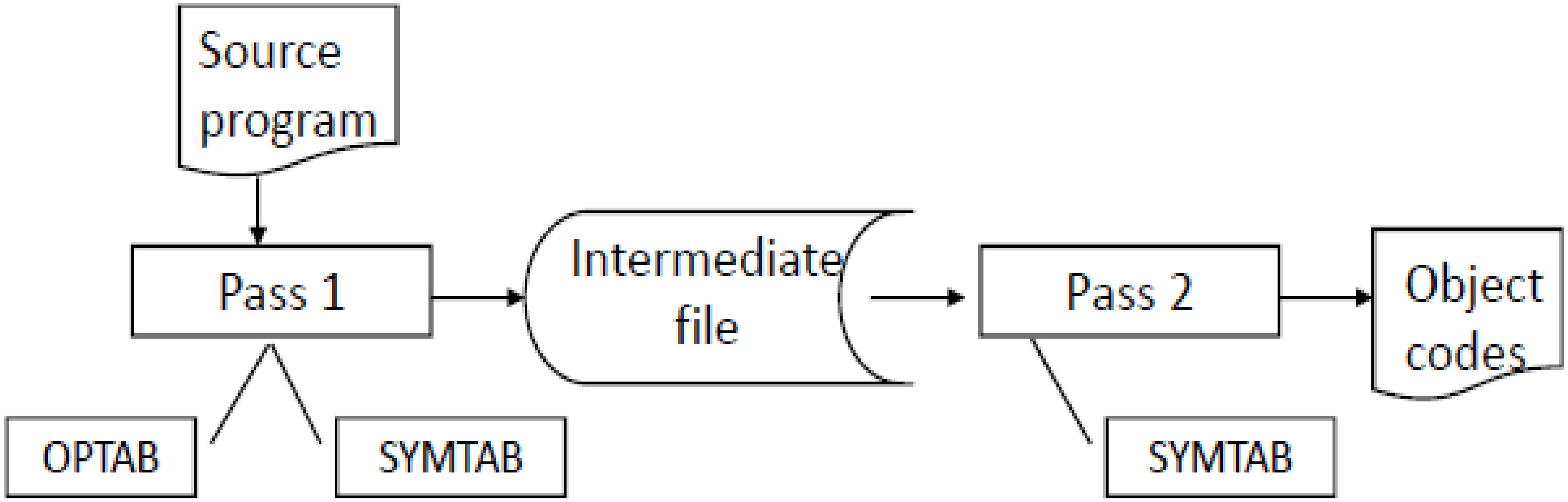
Two Pass Assembler

- Pass 1
  - Assign addresses to all statements in the program
  - Save the values assigned to all labels for use in Pass 2
  - Perform some processing of assembler directives
- Pass 2
  - Assemble instructions
  - Generate data values defined by BYTE, WORD
  - Perform processing of assembler directives not done in Pass 1
  - Write the object program and the assembly listing

Two Pass Assembler

Read from input line

LABEL, OPCODE, OPERAND


Source program
Pass 1
OPTAB
SYMTAB
Intermediate file
Pass 2
SYMTAB
Object codes

Data Structures

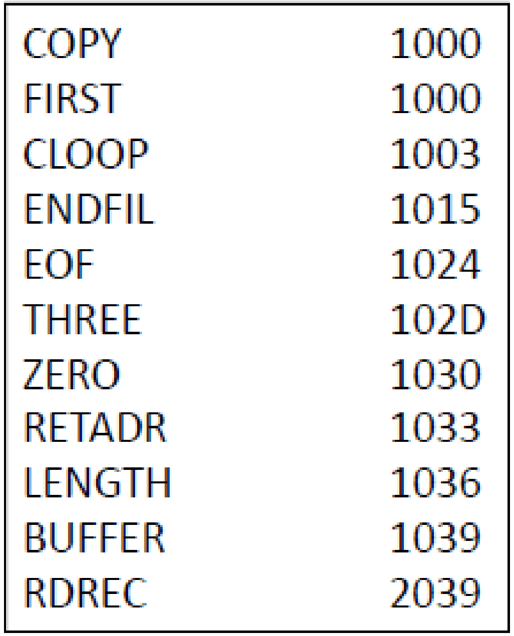
- **Operation Code Table (OPTAB)**
- **Symbol Table (SYMTAB)**
- **Location Counter(LOCCTR)**

Data Structures

- Operation Code Table (OPTAB)
  - Used to lookup mnemonic operation codes and translate them to their machine language equivalents
  - If variable length instruction format: OPTAB contains length of the instruction as well
  - Organized as hash table with mnemonic operation code as key
  - OPTAB is a static table

Data Structures

- Symbol Table (SYMTAB)
  - Used to store values (addresses) assigned to labels
  - Contains the name and value (address) for each label in the source program, together with flags to indicate error conditions
    - label name, value, flag, (type, length) etc
  - Organized as hash table for efficiency of insertion & retrieval
  - Dynamic table (insert, delete, search)

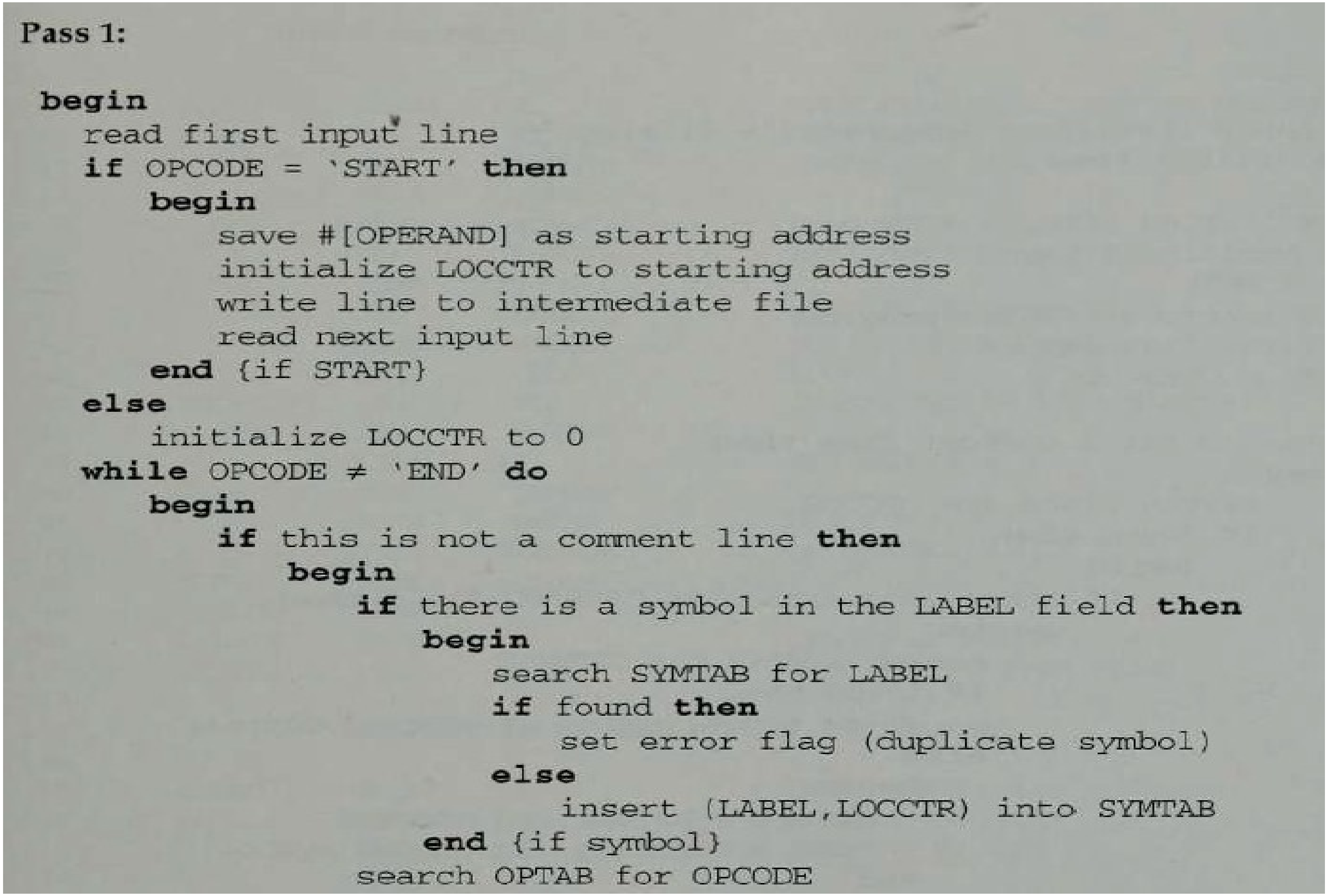


COPY	1000
FIRST	1000
CLOOP	1003
ENDFIL	1015
EOF	1024
THREE	102D
ZERO	1030
RETADR	1033
LENGTH	1036
BUFFER	1039
RDREC	2039

Data Structures

- Location Counter(LOCCTR)
  - Used to help in the assignment of addresses
  - Initialized to the address specified in the START statement
  - After each source statement is processed, the length of the assembled instruction or data area to be generated is added to LOCCTR

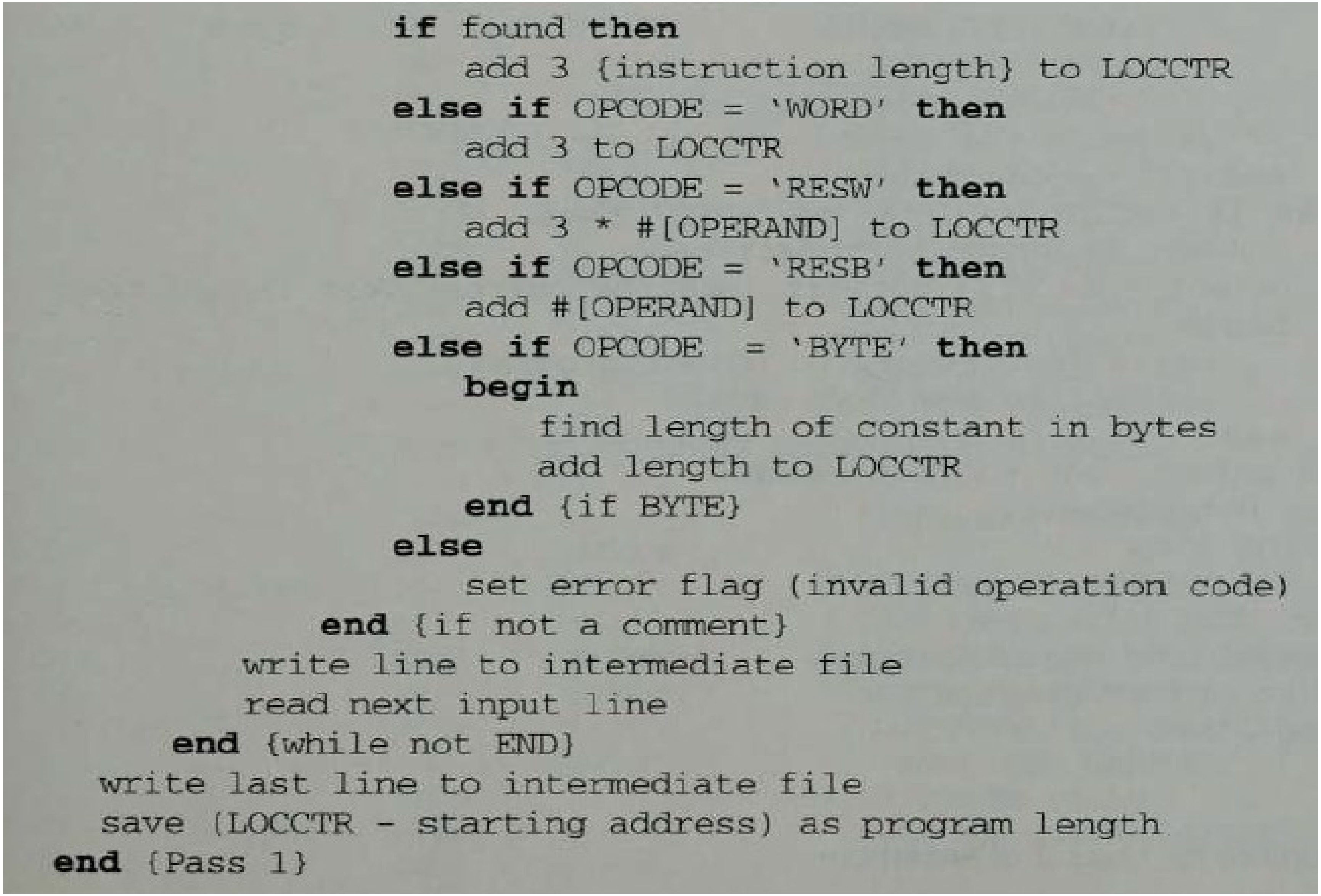
Algorithm for pass1 Assembler



Pass 1:

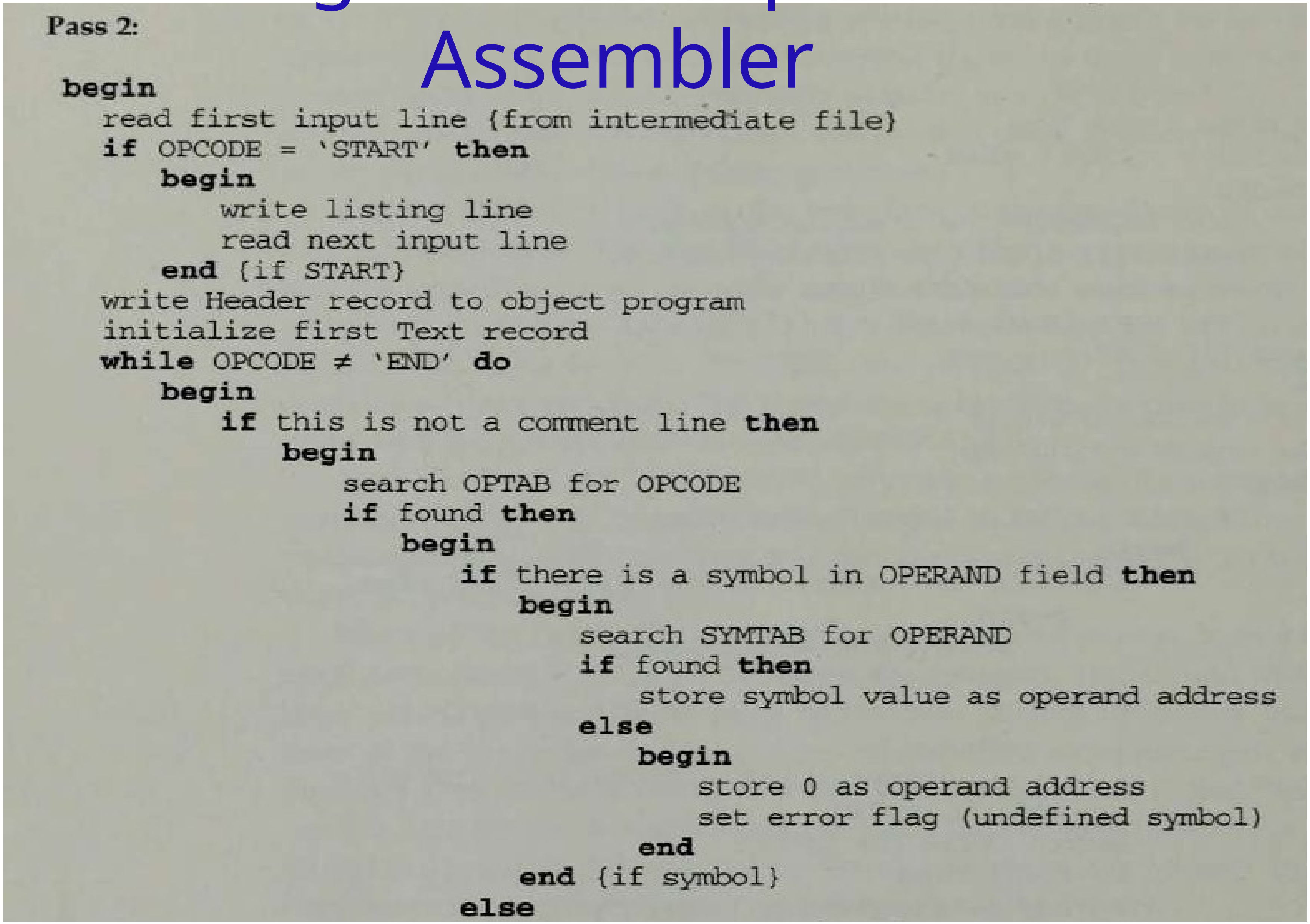
```
begin
   read first input line
   if OPCODE = 'START' then
      begin
         save #[OPERAND] as starting address
         initialize LOCCTR to starting address
         write line to intermediate file
         read next input line
      end {if START}
   else
      initialize LOCCTR to 0
   while OPCODE ≠ 'END' do
      begin
         if this is not a comment line then
            begin
               if there is a symbol in the LABEL field then
                  begin
                     search SYMTAB for LABEL
                     if found then
                        set error flag (duplicate symbol)
                     else
                        insert (LABEL,LOCCTR) into SYMTAB
                  end {if symbol}
               search OPTAB for OPCODE
```

Algorithm for pass1 Assembler



```
                    if found then
                        add 3 {instruction length} to LOCCTR
                    else if OPCODE = 'WORD' then
                        add 3 to LOCCTR
                    else if OPCODE = 'RESW' then
                        add 3 * #[OPERAND] to LOCCTR
                    else if OPCODE = 'RESB' then
                        add #[OPERAND] to LOCCTR
                    else if OPCODE  = 'BYTE' then
                        begin
                            find length of constant in bytes
                            add length to LOCCTR
                        end {if BYTE}
                    else
                        set error flag (invalid operation code)
                end {if not a comment}
            write line to intermediate file
            read next input line
        end {while not END}
    write last line to intermediate file
    save (LOCCTR - starting address) as program length
end {Pass 1}
```

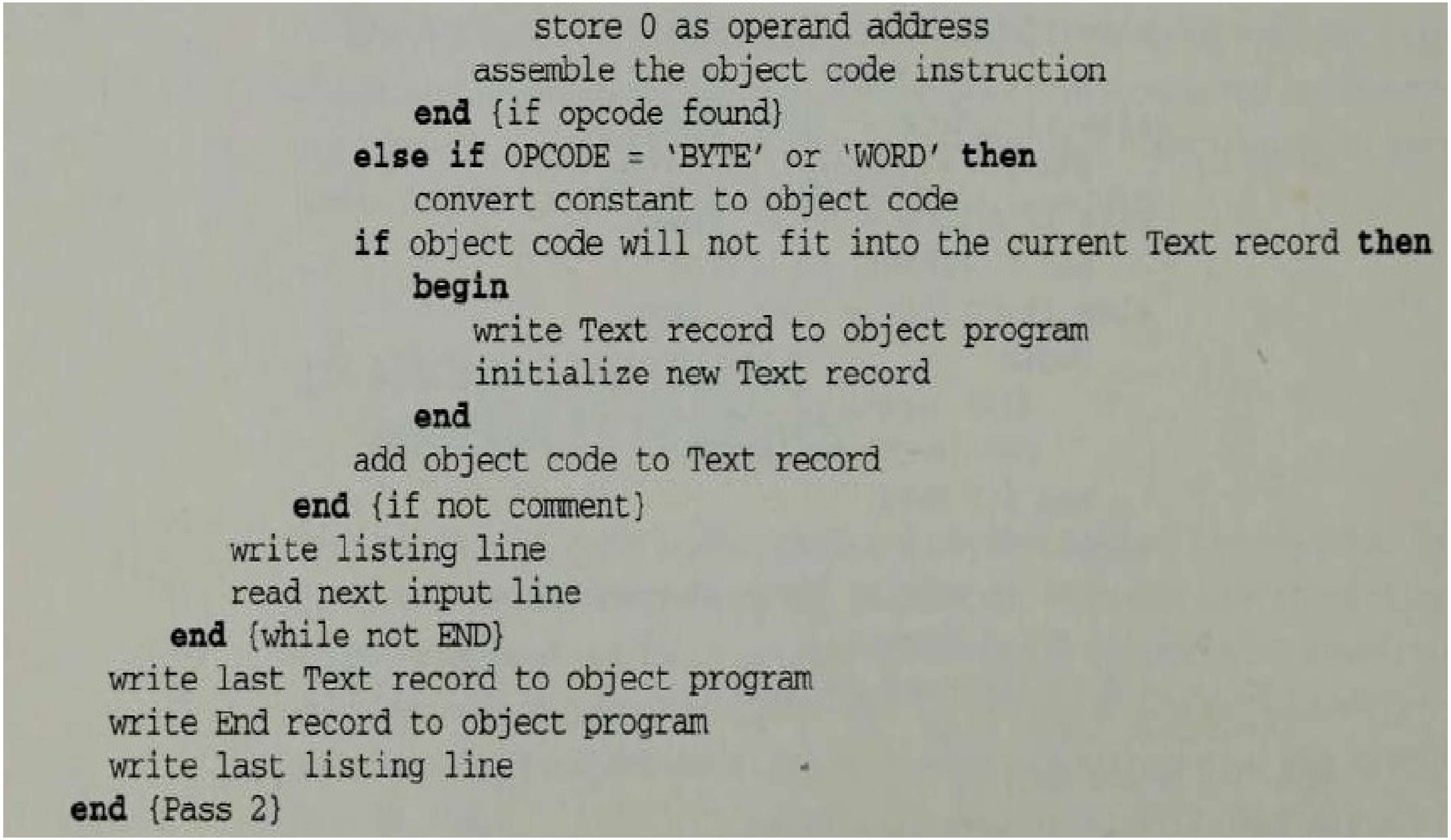
Algorithm for pass 2 Assembler



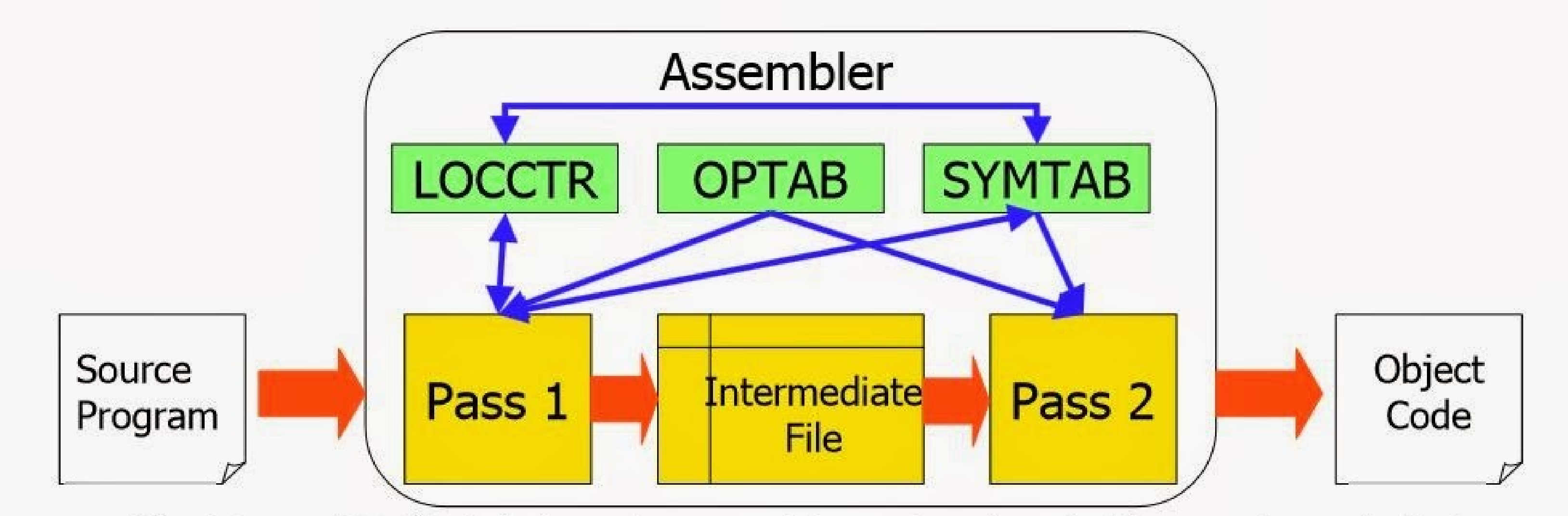
Pass 2:

```
begin
    read first input line {from intermediate file}
    if OPCODE = 'START' then
        begin
            write listing line
            read next input line
        end {if START}
    write Header record to object program
    initialize first Text record
    while OPCODE ≠ 'END' do
        begin
            if this is not a comment line then
                begin
                    search OPTAB for OPCODE
                    if found then
                        begin
                            if there is a symbol in OPERAND field then
                                begin
                                    search SYMTAB for OPERAND
                                    if found then
                                        store symbol value as operand address
                                    else
                                        begin
                                            store 0 as operand address
                                            set error flag (undefined symbol)
                                        end
                                end {if symbol}
                            else
```

Algorithm for pass 2 Assembler



```
                       store 0 as operand address
                    assemble the object code instruction
                 end {if opcode found}
              else if OPCODE = 'BYTE' or 'WORD' then
                 convert constant to object code
              if object code will not fit into the current Text record then
                 begin
                    write Text record to object program
                    initialize new Text record
                 end
              add object code to Text record
           end {if not comment}
        write listing line
        read next input line
     end {while not END}
  write last Text record to object program
  write End record to object program
  write last listing line
end {Pass 2}
```


Assembler
LOCCTR
OPTAB
SYMTAB
Source Program
Pass 1
Intermediate File
Pass 2
Object Code

The intermediate file include each source statement, assigned address and error indicator

OPTAB	
Mnemonic	Opcode
LDX	04
LDA	00
ADD	18
STA	0C
RSUB	4C
TIX	2C
JLT	38

SYMTAB	
Name	LOCCTR
FIRST	1000
LOOP	1006
TABLE	1015
COUNT	1018
ZERO	101B
TOTAL	101E

Assembler Design

- Machine Dependent Assembler Features
  - instruction formats and addressing modes
  - program relocation
- Machine Independent Assembler Features
  - literals
  - symbol-defining statements
  - expressions
  - program blocks
  - control sections and program linking

Machine-dependent Assembler Features

- ✓ Instruction formats and addressing modes
- ✓ Program relocation

Instruction Format and Addressing Mode

- SIC/XE
  - PC-relative or Base-relative addressing: op m
  - Indirect addressing: op @m
  - Immediate addressing: op #c
  - Extended format: +op m
  - Index addressing: op m,x
  - register-to-register instructions
  - larger memory -> multi-programming (program allocation)

Translation

- Register translation
  - register name (A, X, L, B, S, T, F, PC, SW) and their values (0,1, 2, 3, 4, 5, 6, 8, 9)
  - preloaded in SYMTAB
- Address translation
  - Most register-memory instructions use program counter relative or base relative addressing
  - Format 3: 12-bit address field
    - base-relative: 0~4095
    - pc-relative: -2048~2047
  - Format 4: 20-bit address field

Assembler Design Options

- One-pass assemblers
- Multi-pass assemblers
- Two-pass assembler with overlay structure

Two-Pass Assembler with Overlay Structure

- For small memory
  - pass 1 and pass 2 are never required at the same time
  - three segments
    - root: driver program and shared tables and subroutines
    - pass 1
    - pass 2
  - tree structure
  - overlay program

One-Pass Assemblers

- Main problem
  - forward references
    - data items
    - labels on instructions
- Solution
  - data items: require all such areas be defined before they are referenced
  - labels on instructions: no good solution

One-Pass Assemblers

- Main Problem
  - forward reference
    - data items
    - labels on instructions
- Two types of one-pass assembler
  - load-and-go
    - produces object code directly in memory for immediate execution
  - the other
    - produces usual kind of object code for later execution

Load-and-go Assembler

- Characteristics
  - Useful for program development and testing
  - Avoids the overhead of writing the object program out and reading it back
  - Both one-pass and two-pass assemblers can be designed as load-and-go.
  - However one-pass also avoids the over head of an additional pass over the source program
  - For a load-and-go assembler, the actual address must be known at assembly time, we can use an absolute program

Forward Reference in One-pass Assembler

- For any symbol that has not yet been defined
    1. omit the address translation
    2. insert the symbol into SYMTAB, and mark this symbol undefined
    3. the address that refers to the undefined symbol is added to a list of forward references associated with the symbol table entry
    4. when the definition for a symbol is encountered, the proper address for the symbol is then inserted into any instructions previous generated according to the forward

Load-and-go Assembler (Cont.)

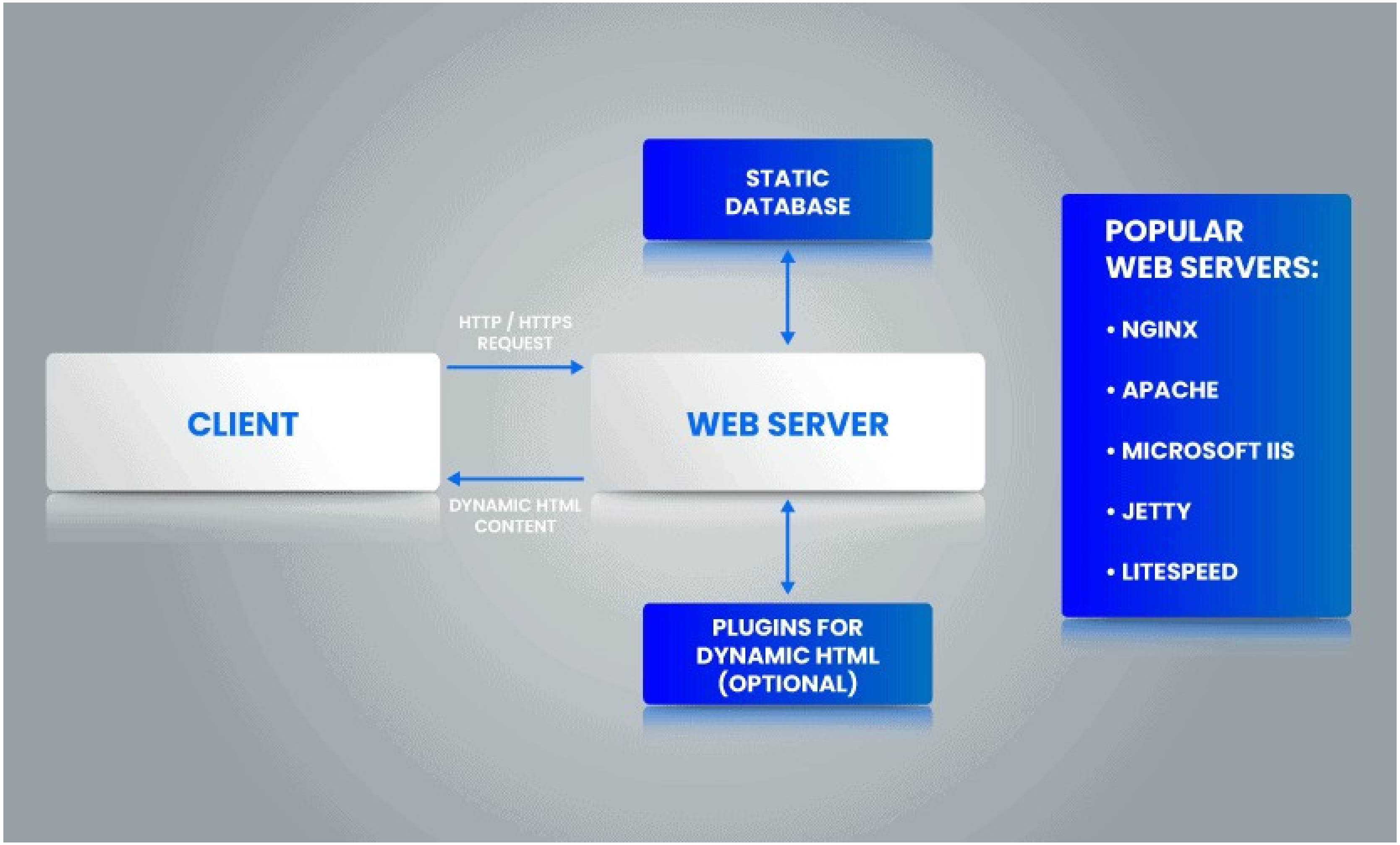
- At the end of the program
  - any SYMTAB entries that are still marked with * indicate undefined symbols
  - search SYMTAB for the symbol named in the END statement and jump to this location to begin execution
- The actual starting address must be specified at assembly time

Producing Object Code

- When external working-storage devices are not available or too slow (for the intermediate file between the two passes
- Solution:
  - When definition of a symbol is encountered, the assembler must generate another Tex record with the correct operand address
  - The loader is used to complete forward references that could not be handled by the assembler
  - The object program records must be kept in their original order when they are presented to the loader

Web Server

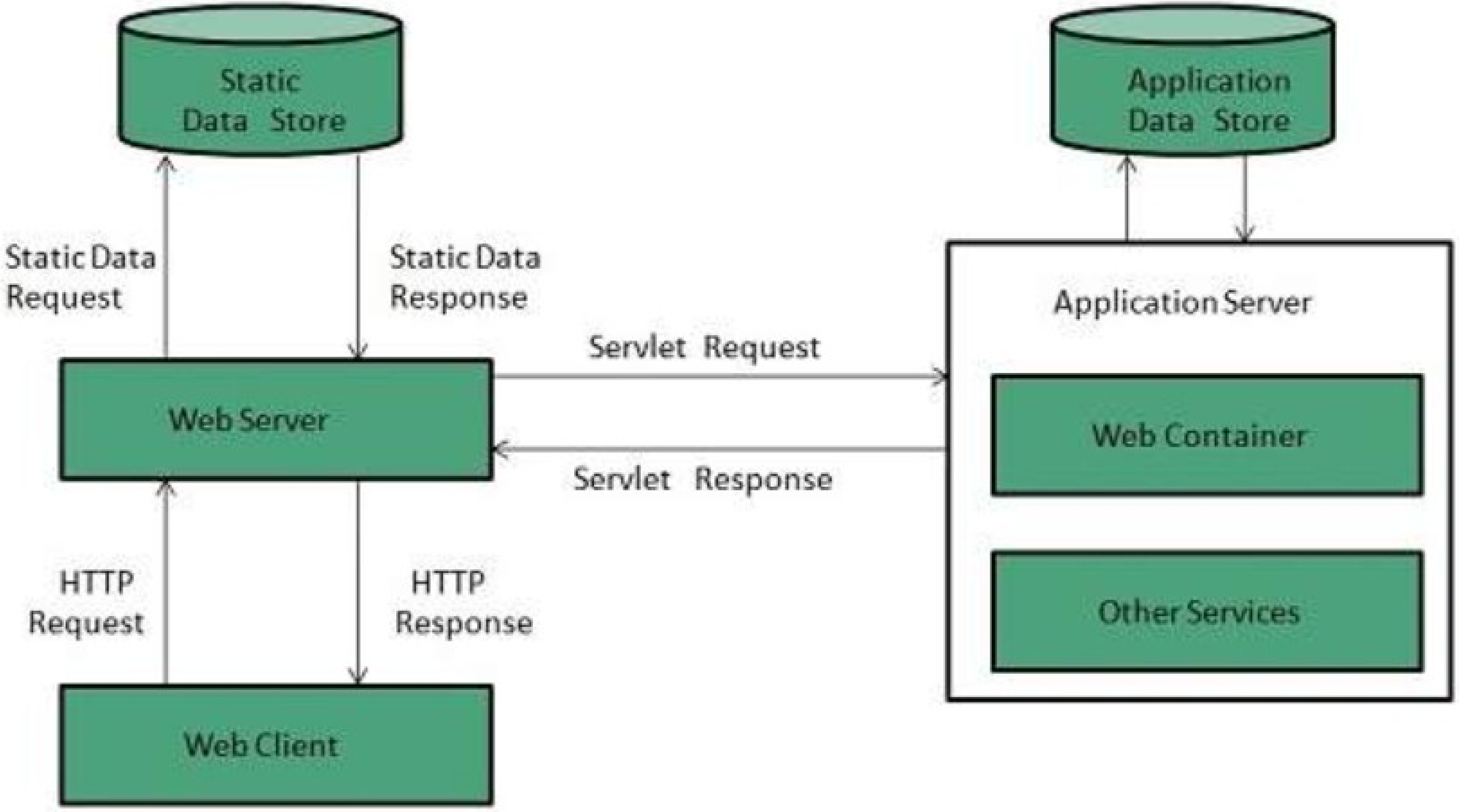
- Web servers are where web content is stored and kept available for users to access at any time.
- The basic objective is to store, process and deliver web pages to the users.
- Only delivers static content via HTML; A hypertext document that displays information on a browser.


STATIC DATABASE
HTTP / HTTPS REQUEST
CLIENT
WEB SERVER
DYNAMIC HTML CONTENT
PLUGINS FOR DYNAMIC HTML (OPTIONAL)

POPULAR WEB SERVERS:

- NGINX
- APACHE
- MICROSOFT IIS
- JETTY
- LITESPEED

Architecture of Web Server


Static Data Store
Application Data Store
Static Data Request
Static Data Response
Application Server
Servlet Request
Web Server
Web Container
Servlet Response
HTTP Request
HTTP Response
Other Services
Web Client

Dynamic vs Static Web Servers

- **A Static Web Server** - consist of a computer and HTTP software, the sever will send hosted files as is to a browser.
- **Dynamic Web Server** - consist of a web server and other software such as an application server and database; the application server can be used to update any hosted files before they are sent to a browser.

Top Web Server Software on the Market

- Apache HTTP Server
- Microsoft Internet Information Services (IIS)
- Nginx
- Lighttpd
- Sun Java System Web Server

Challenges in Webserver Design

- User Interface And User Experience
- Scalability
- Performance
- Knowledge of Framework and Platforms
- Security

Web Server Security Practices

- A reverse proxy is designed to hide an internal server and act as an intermediary for traffic originating on an internal server.
- Access restriction through processes like limiting the web host's access to infrastructure machines or using Secure Socket Shell (SSH).
- Keeping web servers patched and up to date to help ensure the web server isn't susceptible to vulnerabilities.
- Network monitoring to make sure there isn't any or unauthorized activity.
- Using a firewall and SSL as firewalls can monitor HTTP traffic while having a Secure Sockets Layer (SSL) can help keep data secure.

- System calls related to process
  - fork
  - wait
  - exec
  - exit
  - signal
  - kill
  - raise

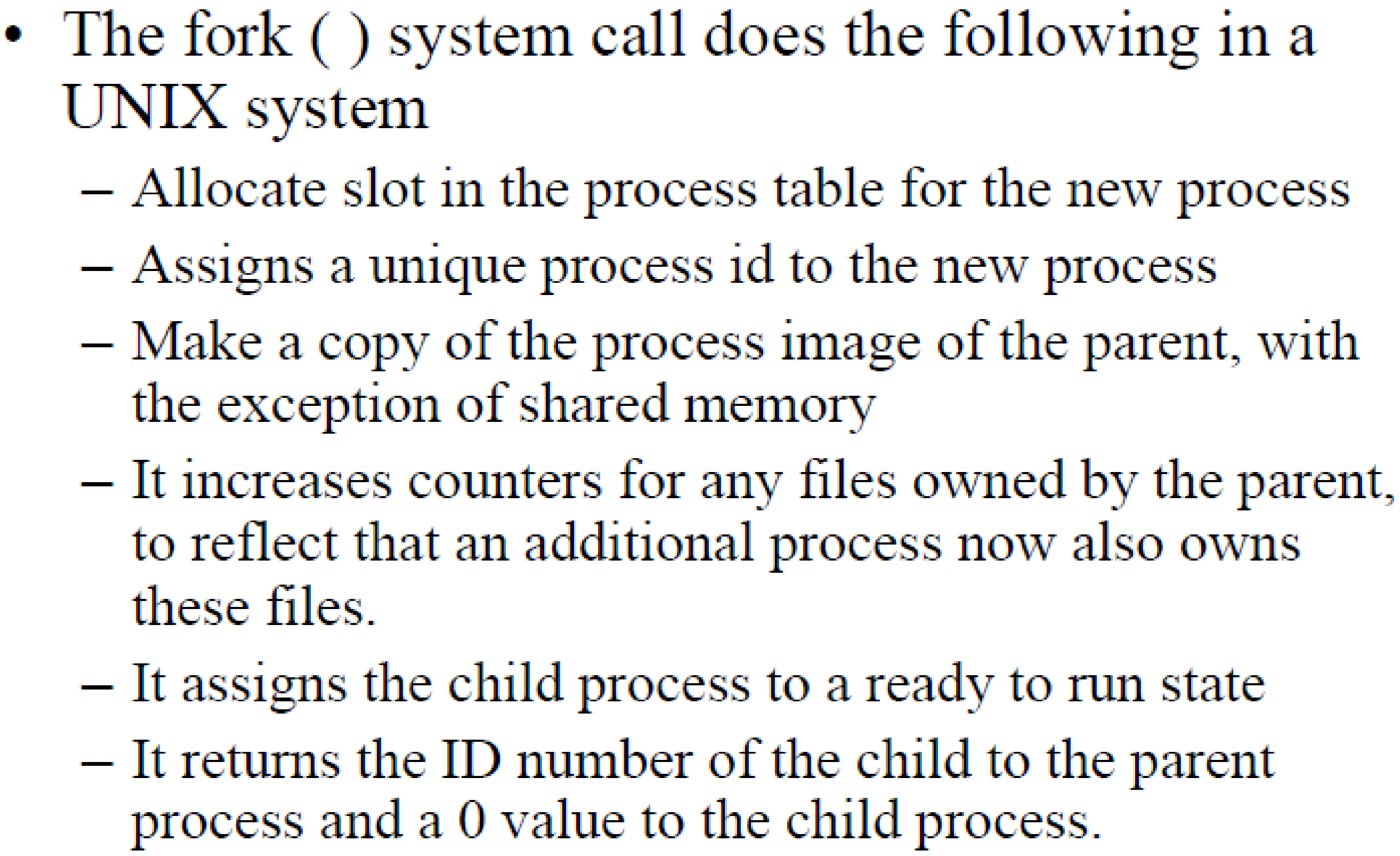
Process summary

- Process is a program in execution
  - Program is passive entity and Process is active entity
- In multi programming / multi tasking environment number of processes can reside in the system
- At any instance of time processes can be in different state such as Ready, Blocked, Running etc. and processes move from one state to another state depending upon certain conditions
- Processes use available system resources
- OS is responsible for managing all the processes in the system

Process related system calls – fork

- pid_t fork(void)
  - Duplicates the current process, creating a new entry in the process table with many of the same attributes as the parent process.
  - The new process is almost identical to the original, executing the same code but with its own data space, environment and file descriptor.
  - Uses copy on write technique
  - Kernel internally invokes do_fork() function (in fork.c)
  - do_fork(SIGCHLD, regs.esp, &regs, 0);

Process related system calls – fork

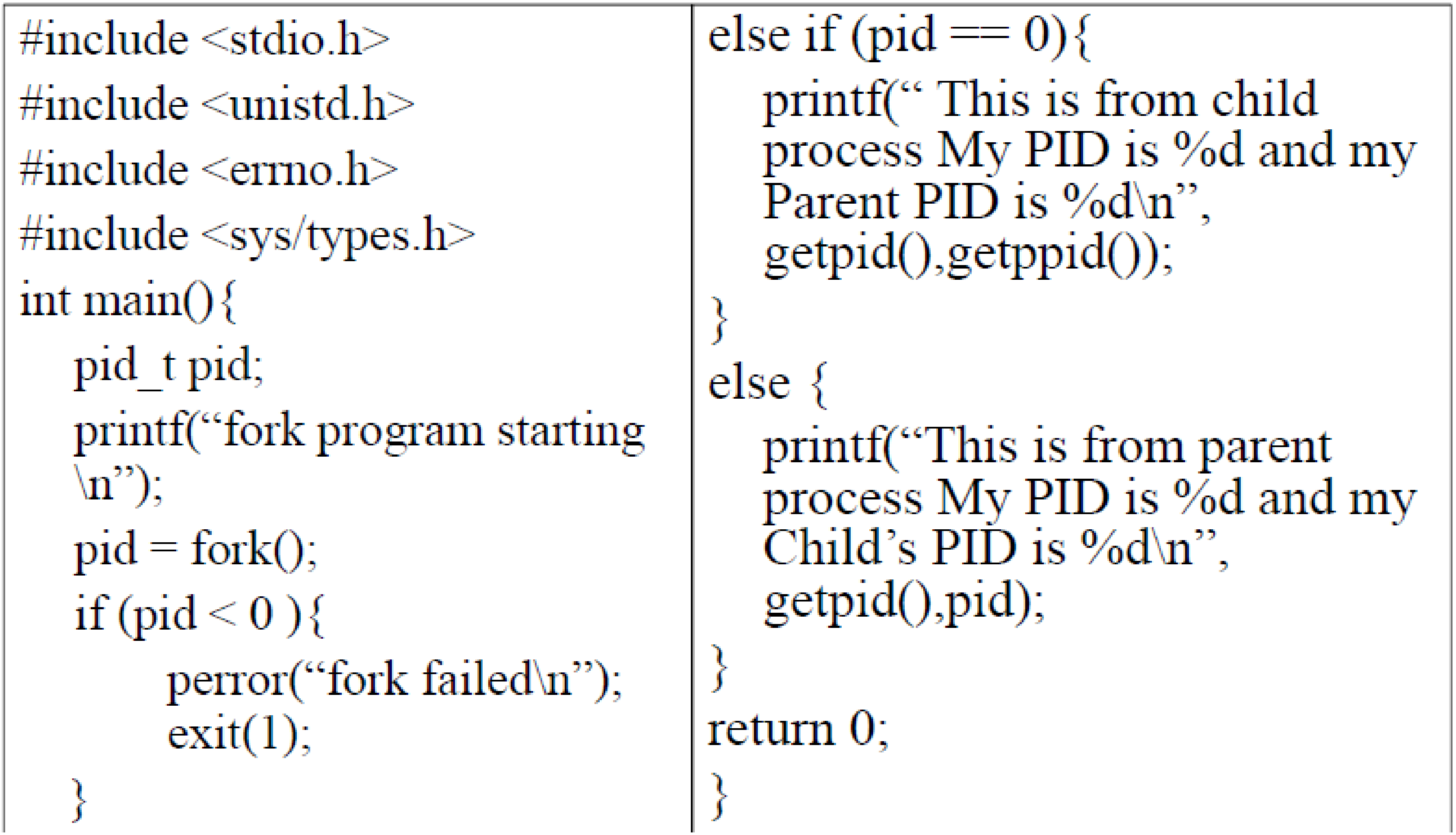


- The fork ( ) system call does the following in a UNIX system
    - – Allocate slot in the process table for the new process
    - – Assigns a unique process id to the new process
    - – Make a copy of the process image of the parent, with the exception of shared memory
    - – It increases counters for any files owned by the parent, to reflect that an additional process now also owns these files.
    - – It assigns the child process to a ready to run state
    - – It returns the ID number of the child to the parent process and a 0 value to the child process.

Process related system calls – fork

  - All these works is done in Kernel space of parent process.
  - After completing these functions, OS will do the following operations as a part of dispatcher routine
  - Control returns to the user mode at the point of the fork call of the parent.
  - Transfer control to the child process. The child process begins executing at the same point in the code as the parent, namely at the return from the fork call.
  - Transfer control to another process. Both child and parent are left in the ready to run state.
- If fork system call fails, the return value to parent (no child will be created) will be -1.

Process Creation Example



```
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
int main(){
    pid_t pid;
    printf(“fork program starting
    \n”);
    pid = fork();
    if (pid < 0 ){
        perror(“fork failed\n”);
        exit(1);
    }
```

```
else if (pid == 0){
    printf(“ This is from child
    process My PID is %d and my
    Parent PID is %d\n”,
    getpid(),getppid());
}
else {
    printf(“This is from parent
    process My PID is %d and my
    Child’s PID is %d\n”,
    getpid(),pid);
}
return 0;
}
```

Calling fork()

Calling fork()

Consider the following example in which we have used the fork() system call to create a new child process:

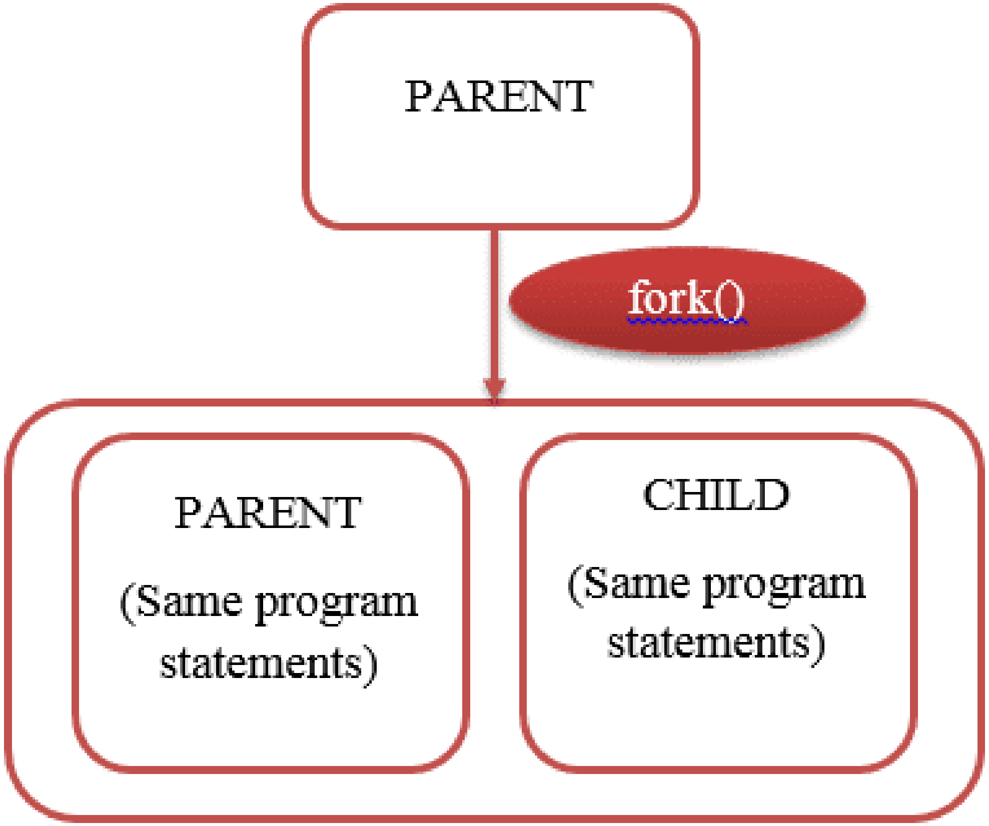
```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
   fork();
   printf("Using fork() system call\n");
   return 0;
}
```

OUTPUT:

```
sysads@linuxhint $ gcc fork.c -o fork
sysads@linuxhint $ ./fork

Using fork() system call
Using fork() system call
```

In this program, we have used fork(), this will create a new child process. When the child process is created, both the parent process and the child process will point to the next instruction (same Program Counter).

That is 2^n times, where n is the number of fork() system calls.



PARENT

fork()

PARENT

(Same program statements)

CHILD

(Same program statements)

Calling fork()

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
   fork();
   fork();
   fork();
   fork();
   printf("Using fork() system call\n");
   return 0;
}
```

Output:

```
sysads@linuxhint $ gcc fork.c -o fork
sysads@linuxhint $ ./fork
```

Now the total number of process created are $2^4 = 16$ and we have our print statement executed 16 times.

Calling fork()

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
   pid_t p;
   p = fork();
   if(p==-1)
   {
     printf("There is an error while calling
fork()\n");
   }
   if(p==0)
   {
      printf("We are in the child process\n");
   }
   else
   {
      printf("We are in the parent process\
n");
   }
   return 0;
```

OUTPUT:

```
sysads@linuxhint $ gcc fork.c -o
fork
sysads@linuxhint $ ./fork
We are in the parent process
We are in the child process
```

Calling fork()

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
//main function begins
int main(){
   pid_t p= fork(); //calling of fork system call
   if(p==-1)
      printf("Error occured while calling fork()");
   else if(p==0)
      printf("This is the child process");
   else
      printf("This is the parent process");
   return 0;
}
```

Output:

```
This is the parent process

This is the child process
```

THANK YOU

BITS Pilani

Pilani Campus

BITS Pilani
Pilani Campus

COMPUTING AND DESIGN
SESSION 15

Course design by - Dr. Lucy J. Gudino &
Dr Chandrasekhar
Faculty – Dr Thangakumar J

BITS Pilani
Pilani Campus

Operating Systems – Abstraction vs. Realization

Last Session

List of Topic Title	Text/Ref Book/external resource
• Inheritance and Delegation; Design with inheritance and delegation; Comparison: Program Composition – Object Hierarchy vs. Class Hierarchy • OO Design and Design Patterns • Design of Complex Software (Compilers/ OS/ Web Server as an example).	R4

OS: Abstraction vs. Realization

Today's topic

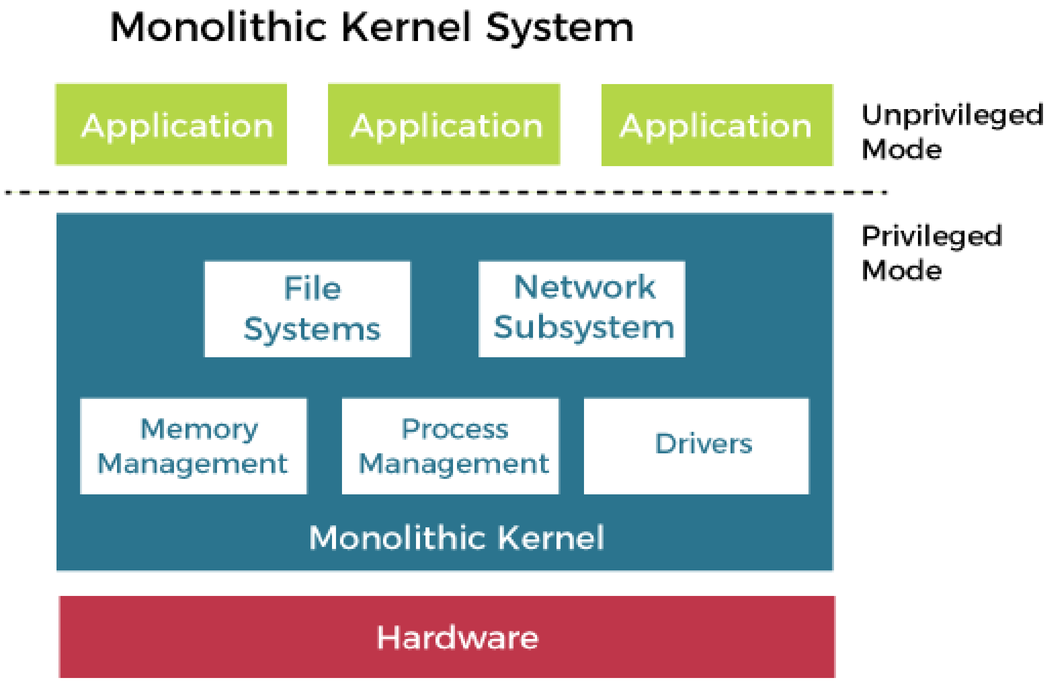
Operating System Structure

- Monolithic Systems
- Layered Systems
- Microkernels
- Client Server Model
- Virtual Machines

Operating System Structure

- **The six designs are:**
  - Monolithic systems,
  - Layered systems,
  - Microkernels,
  - Client-server systems,
  - Virtual machines, and
  - Exokernels

Monolithic Systems


Monolithic Kernel System
Application
Application
Application
Unprivileged Mode
Privileged Mode
File Systems
Network Subsystem
Memory Management
Process Management
Drivers
Monolithic Kernel
Hardware

Monolithic Systems

- The entire OS runs as a single program in kernel mode
- The OS - a collection of procedures, linked together into a single large executable binary program
- Each procedure in the system is free to call any other one; efficient but leads to a system unwieldy and difficult to understand
- A crash in any of these procedures will take down the entire OS

- It is possible to have some structure.
- The services (system calls) provided by the OS are requested by putting the parameters in a well-defined place (e.g., on the stack) and then executing a trap instruction.
- This instruction switches the machine from user mode to kernel mode and transfers control to the OS as in figure.
- The OS then fetches the parameters and determines which system call is to be carried out.
- It indexes into a table that contains in slot k a pointer to the procedure that carries out system call k.

A basic structure for the operating system:

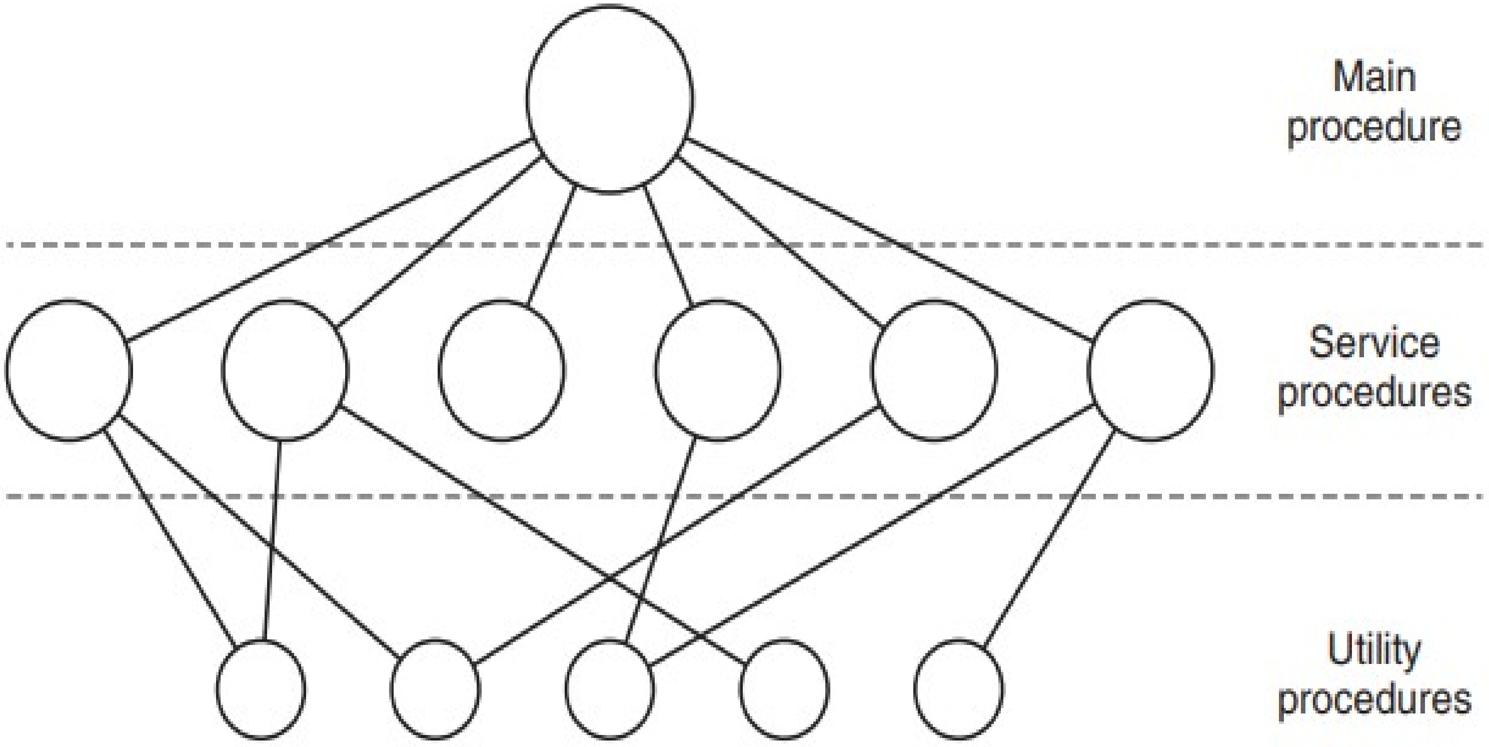
1. A main program that invokes the requested service procedure.

2. A set of service procedures that carry out the system calls.

3. A set of utility procedures that help the service procedures.

- For each system call, there is one service procedure that takes care of it and executes it.
- The utility procedures do things that are needed by several service procedures, such as fetching data from user programs.
- This division of the procedures into three layers is shown in figure.

A Simple Structuring Model for a Monolithic System

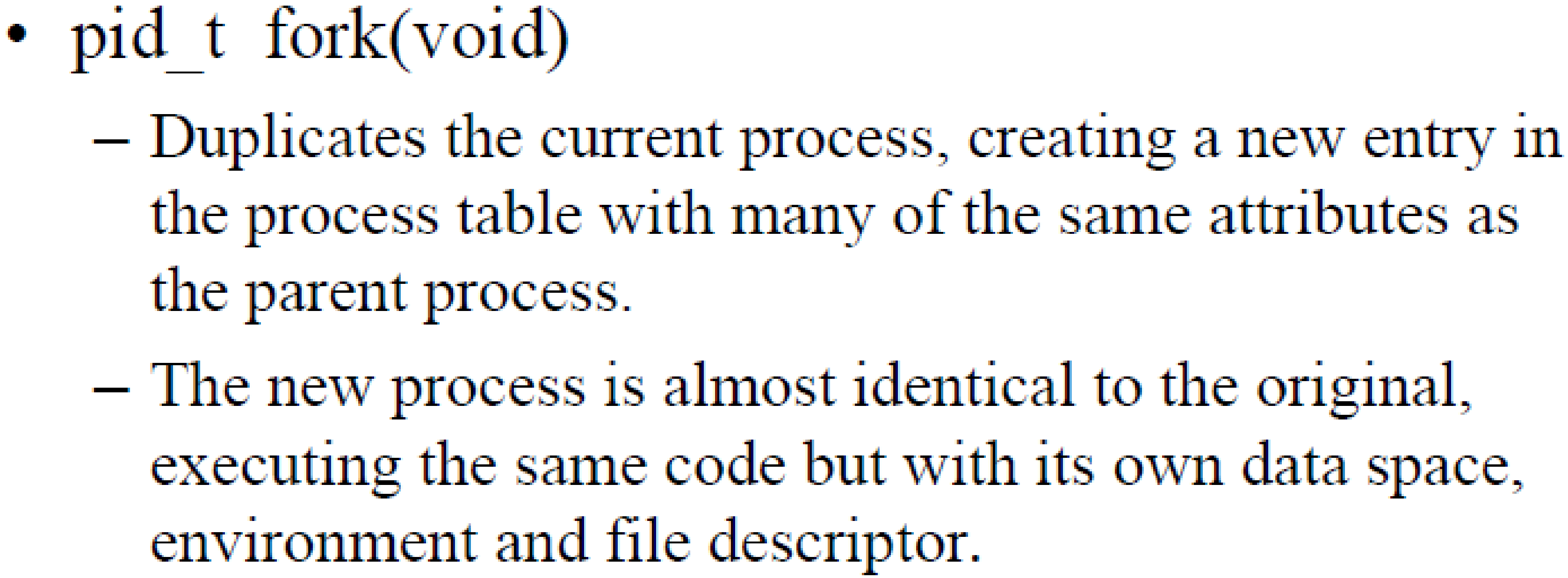

Main procedure
Service procedures
Utility procedures

- System calls related to process
  - fork
  - wait
  - exec
  - exit
  - signal
  - kill
  - raise

Process summary

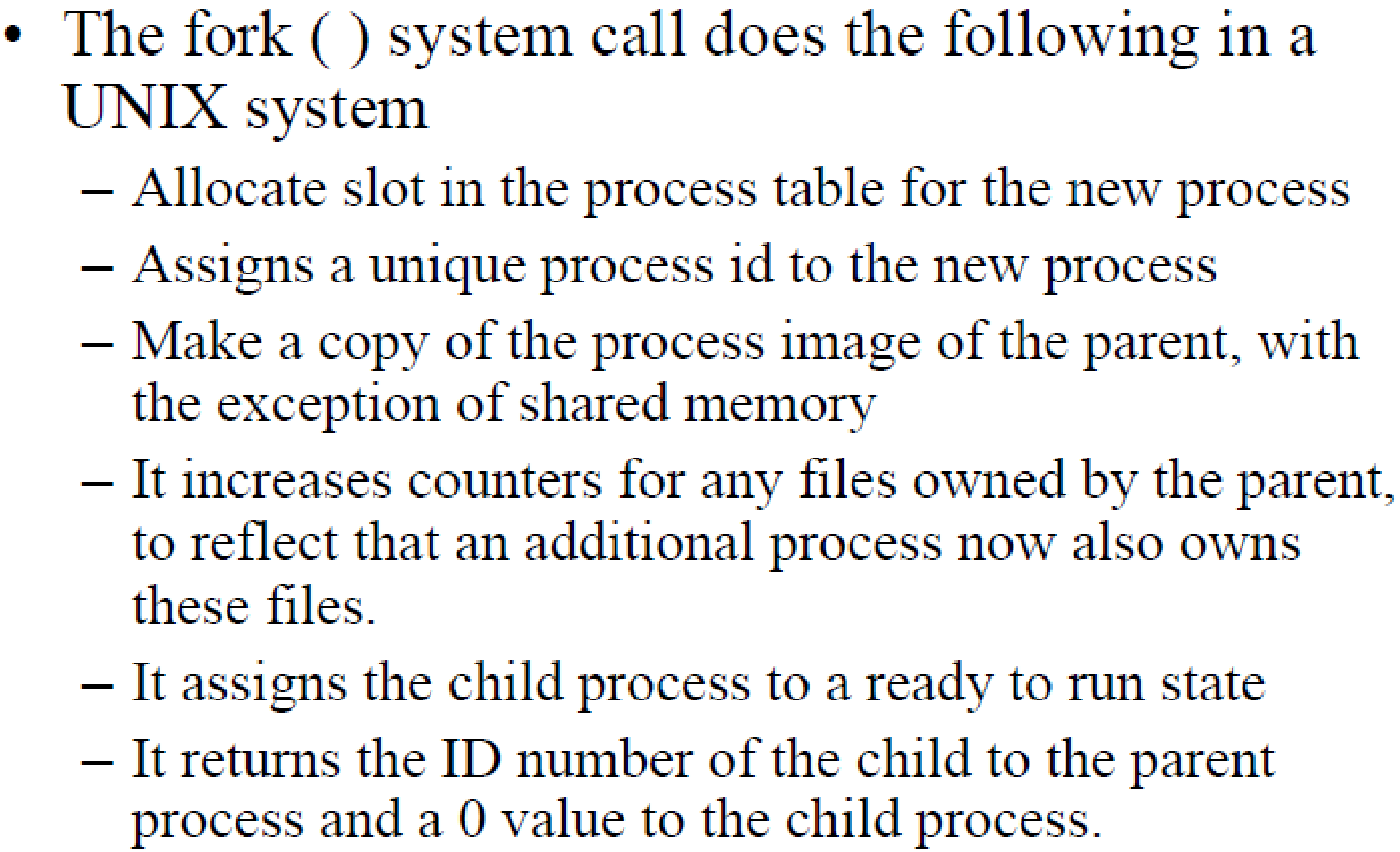
- Process is a program in execution
  - Program is passive entity and Process is active entity
- In multi programming / multi tasking environment number of processes can reside in the system
- At any instance of time processes can be in different state such as Ready, Blocked, Running etc. and processes move from one state to another state depending upon certain conditions
- Processes use available system resources
- OS is responsible for managing all the processes in the system

Process related system calls – fork



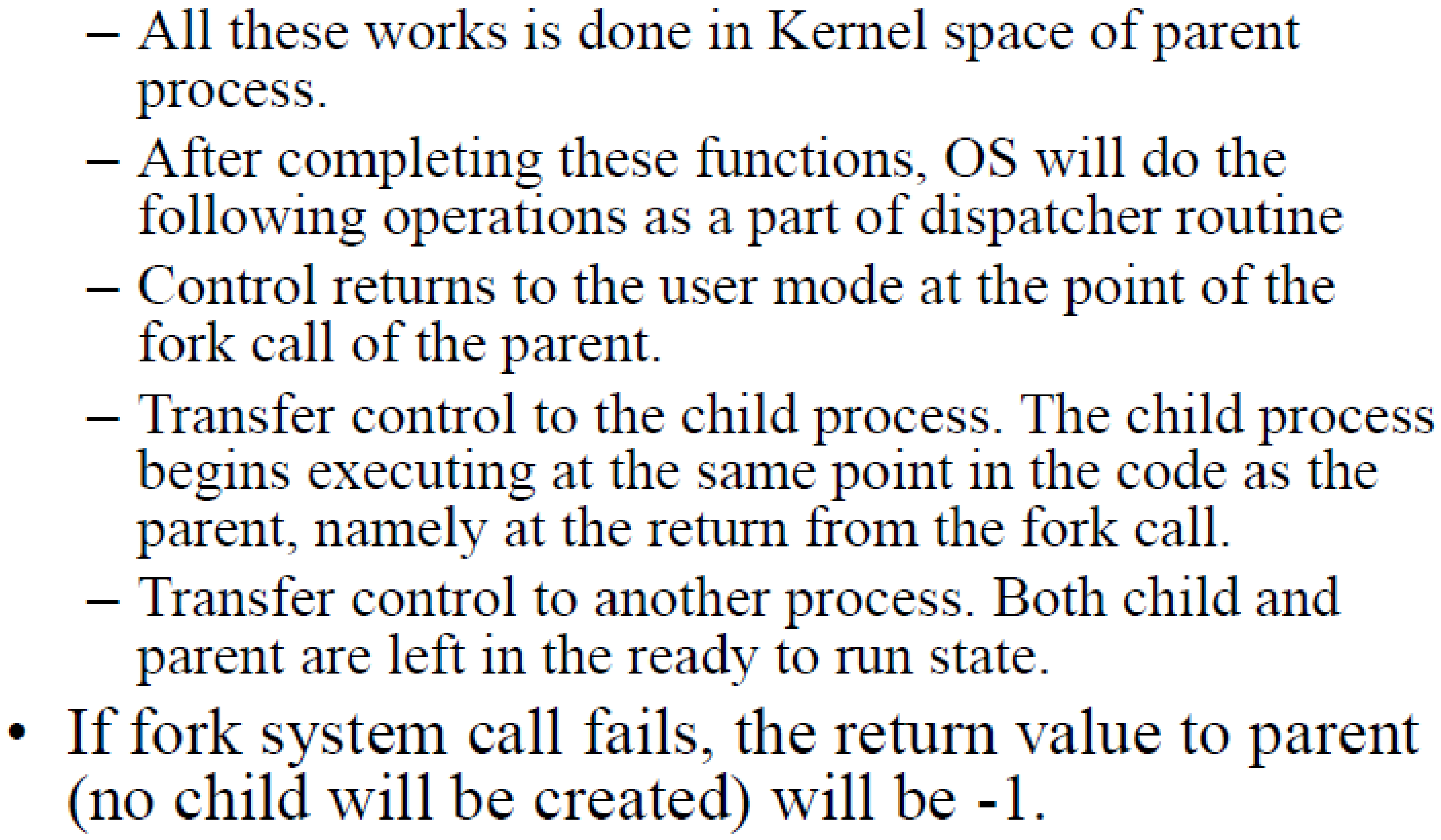
- pid_t fork(void)
  - Duplicates the current process, creating a new entry in the process table with many of the same attributes as the parent process.
  - The new process is almost identical to the original, executing the same code but with its own data space, environment and file descriptor.
  - Uses copy on write technique
  - Kernel internally invokes do_fork() function (in fork.c)
  - do_fork(SIGCHLD, regs.esp, &regs, 0);

Process related system calls – fork



- The fork ( ) system call does the following in a UNIX system
  - Allocate slot in the process table for the new process
  - Assigns a unique process id to the new process
  - Make a copy of the process image of the parent, with the exception of shared memory
  - It increases counters for any files owned by the parent, to reflect that an additional process now also owns these files.
  - It assigns the child process to a ready to run state
  - It returns the ID number of the child to the parent process and a 0 value to the child process.

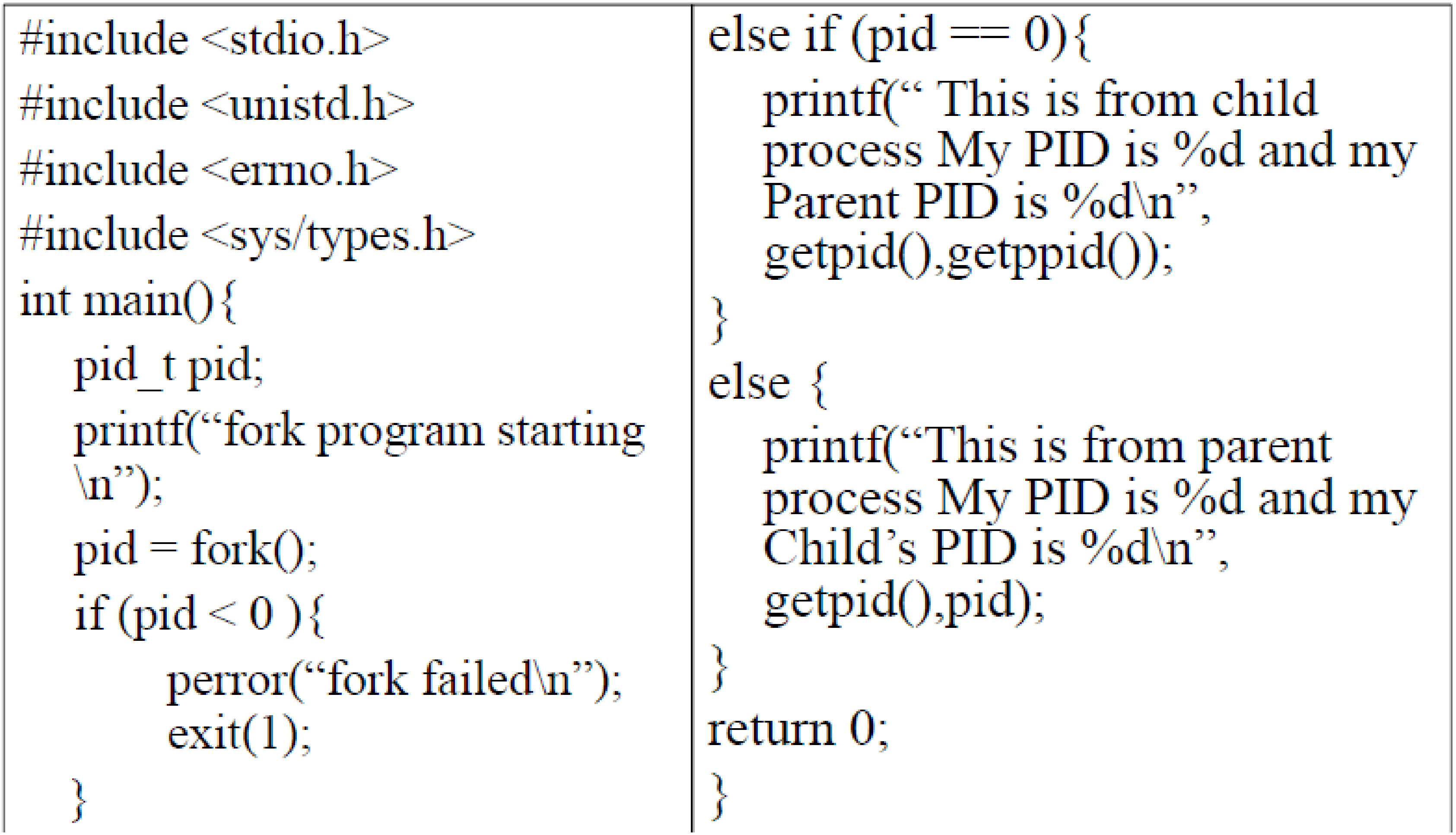
Process related system calls – fork



- All these works is done in Kernel space of parent process.
- After completing these functions, OS will do the following operations as a part of dispatcher routine
- Control returns to the user mode at the point of the fork call of the parent.
- Transfer control to the child process. The child process begins executing at the same point in the code as the parent, namely at the return from the fork call.
- Transfer control to another process. Both child and parent are left in the ready to run state.

- If fork system call fails, the return value to parent (no child will be created) will be -1.

Process Creation Example



```
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
int main(){
    pid_t pid;
    printf(“fork program starting
    \n”);
    pid = fork();
    if (pid < 0 ){
        perror(“fork failed\n”);
        exit(1);
    }
else if (pid == 0){
    printf(“ This is from child
    process My PID is %d and my
    Parent PID is %d\n”,
    getpid(),getppid());
}
else {
    printf(“This is from parent
    process My PID is %d and my
    Child’s PID is %d\n”,
    getpid(),pid);
}
return 0;
}
```

Calling fork()

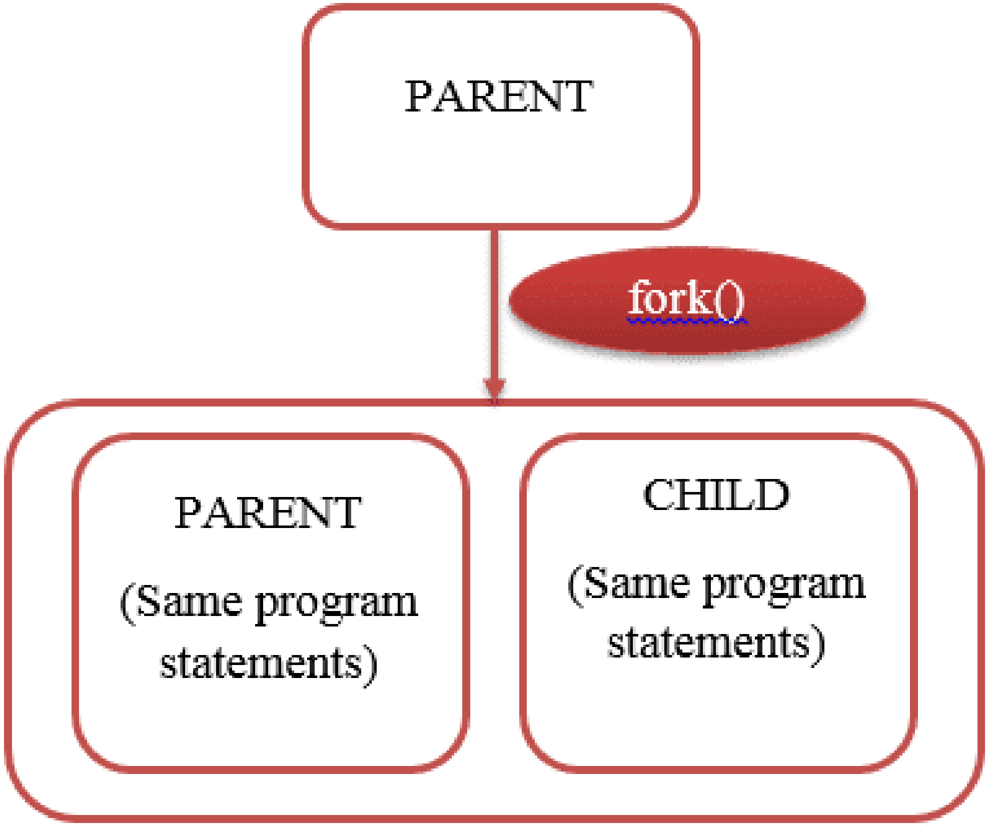
Calling fork()
Consider the following example in which we have used the fork() system call to create a new child process:

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
   fork();
   printf("Using fork() system call\n");
   return 0;
}
```

OUTPUT:

```
sysads@linuxhint $ gcc fork.c -o fork
sysads@linuxhint $ ./fork

Using fork() system call
Using fork() system call
```

In this program, we have used fork(), this will create a new child process. When the child process is created, both the parent process and the child process will point to the next instruction (same Program Counter).
That is 2^n times, where n is the number of fork() system calls.



PARENT

fork()

PARENT

(Same program statements)

CHILD

(Same program statements)

Calling fork()

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
   fork();
   fork();
   fork();
   fork();
   printf("Using fork() system call\
n");
   return 0;
}
```

Output:

```
sysads@linuxhint $ gcc fork.c -
o fork
sysads@linuxhint $ ./fork
```

Now the total number of process created are $2^4 = 16$ and we have our print statement executed 16 times.

Calling fork()

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
   pid_t p;
   p = fork();
   if(p==-1)
   {
     printf("There is an error while calling
fork()\n");
   }
   if(p==0)
   {
      printf("We are in the child process\n");
   }
   else
   {
      printf("We are in the parent process\
n");
   }
   return 0;
```

OUTPUT:

```
sysads@linuxhint $ gcc fork.c -o
fork
sysads@linuxhint $ ./fork
We are in the parent process
We are in the child process
```

Calling fork()

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
//main function begins
int main(){
   pid_t p= fork(); //calling of fork system call
   if(p==-1)
      printf("Error occured while calling fork()");
   else if(p==0)
      printf("This is the child process");
   else
      printf("This is the parent process");
   return 0;
}
```

Output:

```
This is the parent process

This is the child process
```

Advantages

1. The monolithic kernel runs quickly because of memory management, file management, process scheduling, etc. These are all implemented in the same address space.
2. Its structures are easy and simple. The kernel contains all of the components required for processing.
3. All of the components may interact directly with each other's and also with the kernel.
4. It works better for performing smaller tasks because it may handle limited resources.

Disadvantages

5. Monolithic OS has more tendency to generate errors and bugs. It's because user programs use the same address spaces as the kernel.
6. It isn't easy to port code written in the monolithic operating system.
7. It is very difficult to add and remove features from a monolithic operating system. All of the code must be modified and recompiled to add or remove a feature.

Layered Systems

- THE system built at the Technische Hogeschool Eindhoven in the Netherlands by E. W. Dijkstra (1968) and his students.
- The system was a simple batch system for a Dutch computer, the Electrologica X8, which had 32K of 27-bit words (bits were expensive back then).
- The system had **six layers**, as shown in figure.
- **Layer 0** - allocation of the processor, switching between processes when interrupts occurred or timers expired
- **Above layer 0** - sequential processes; be programmed without having to worry about the fact that multiple processes were running on a single processor.

Recap - Structure of the operating system

Layer	Function
5	The operator
4	User programs
3	Input/Output Management
2	Operator-process communication
1	Memory and drum management
0	Processor allocation and multiprogramming

- **Layer 1** - memory management
- It allocated space for processes in main memory and on a 512K word drum used for holding parts of processes (pages) for which there was no room in main memory
- **Above layer 1** - processes did not have to worry about whether they were in memory or on the drum;
- **The layer 1 software -** pages were brought into memory at the moment they were needed and removed when they were not needed.
- **Layer 2** - communication between each process and the operator console; each process effectively had its own operator console.

- **Layer 3** - managing the I/O devices and buffering the information streams to and from them.
- **Above layer 3** - each process deal with abstract I/O devices with nice properties.
- **Layer 4** - the user programs were found.
- They did not have to worry about process, memory, console, or I/O management.
- The system operator process was located in **layer 5**.
- **MULTICS** - a series of concentric rings, with the inner ones being more privileged than the outer ones

- When a procedure in an outer ring wanted to call a procedure in an inner ring, it had to make the equivalent of a system call, i.e., a TRAP instruction whose parameters were carefully checked for validity before the call was allowed to proceed.
- Although the entire OS was part of the address space of each user process in MULTICS, the hardware made it possible to designate individual procedures (memory segments, actually) as protected against reading, writing, or executing.
- The **THE layering scheme -** a design aid.
- **Advantage of the ring mechanism** - easily be extended to structure user subsystems

Advantages

1. Easy Debugging

It is very simple to debug because the layers are discrete. If an error happens in the CPU scheduling layer, the developer may only debug that layer.

2. Modularity

This design supports modularity because each layer only executes tasks it is scheduled to perform.

3. Abstraction

Each layer is concerned with its own set of functions. As a result, the functions and implementations of the other layer are abstract to it.

4. Easy update

A modification in one layer does not affect the other layers.

Disadvantages

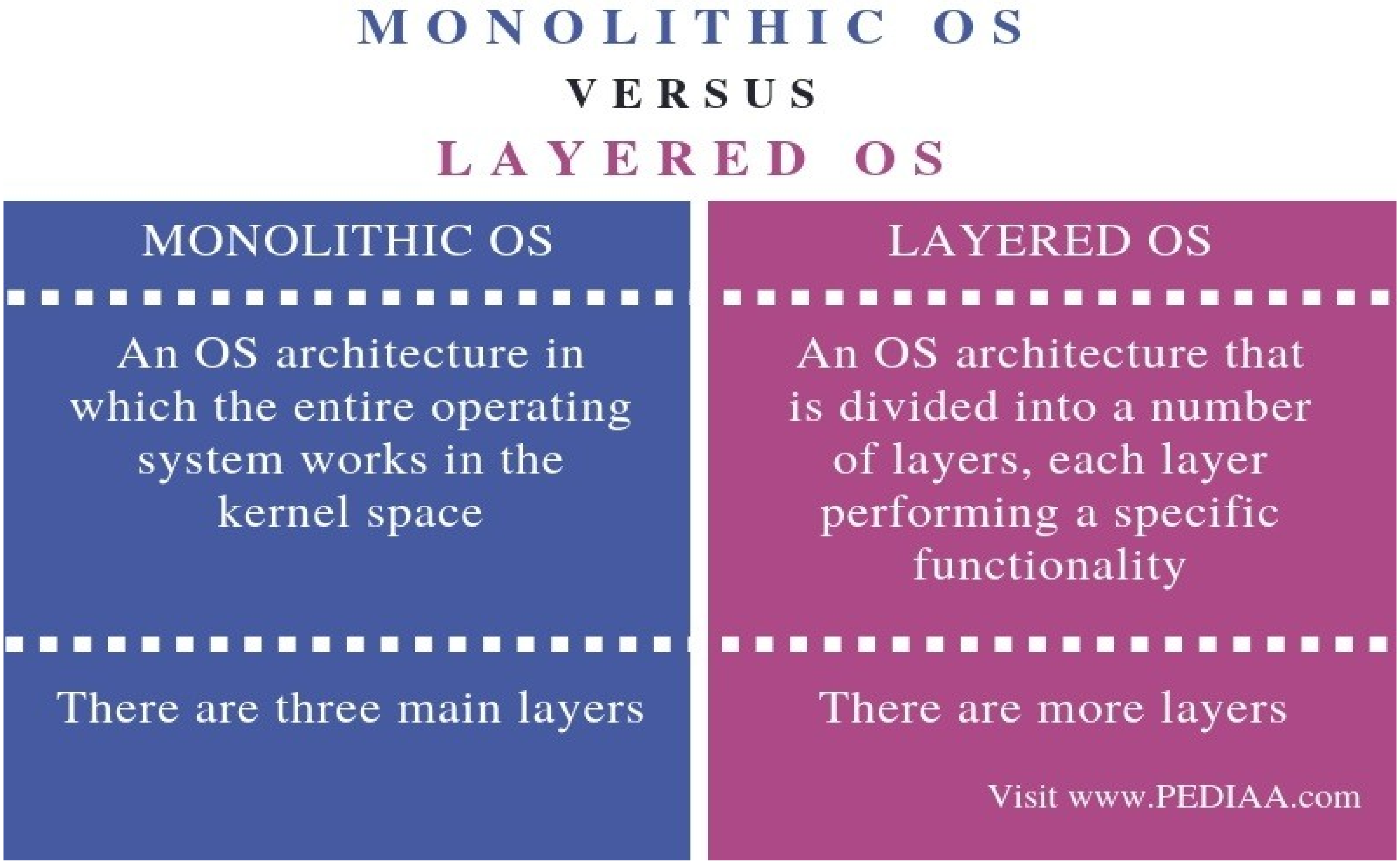
1. Complex and better implementation

Layer layout is important because a layer can utilize the services of the layers below it. For example, the backup storage layer uses the memory management layer's services, so it must be stored beneath the memory management layer.

2. Slower in execution

When one layer wishes to interact with another, it sends a request that must traverse all layers between the two layers to be fulfilled. It enhances response time, which is faster than the Monolithic system. As a result, increasing the number of layers may lead to a very inefficient design.

Summary



MONOLITHIC OS

VERSUS

LAYERED OS

MONOLITHIC OS	LAYERED OS
An OS architecture in which the entire operating system works in the kernel space	An OS architecture that is divided into a number of layers, each layer performing a specific functionality
There are three main layers	There are more layers

Visit www.PEDIAA.com

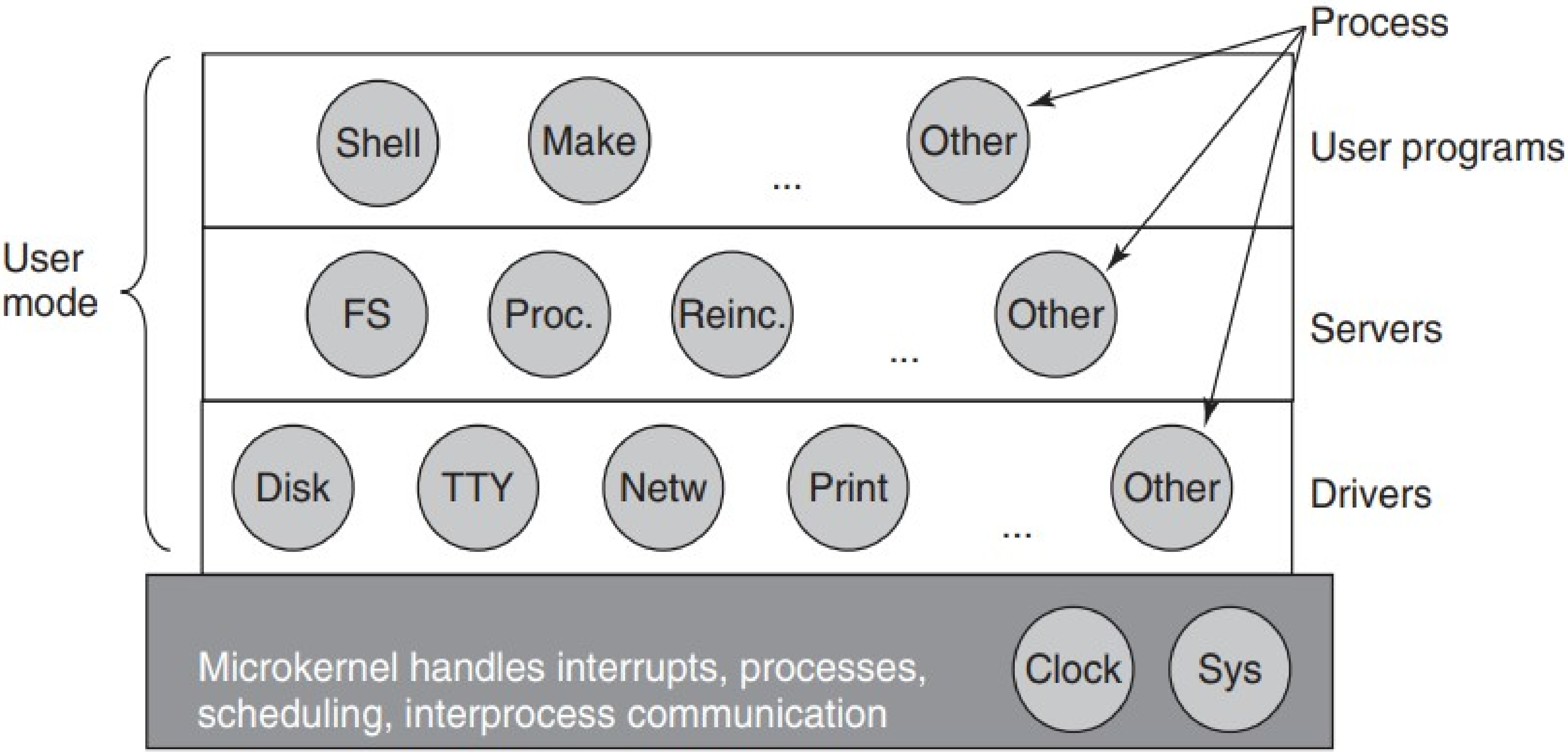
Microkernels

- Various researchers have repeatedly studied the number of bugs per 1000 lines of code
- **Bug density** - module size, module age, and more; but a ballpark figure for serious industrial systems is between two and ten bugs per thousand lines of code.
- A monolithic OS of five million lines of code is likely to contain between 10,000 and 50,000 kernel bugs
- **Microkernel design** - to achieve high reliability by splitting the OS into small, well-defined modules; runs in kernel mode and the rest run as relatively powerless ordinary user processes.

- By running each device driver and file system as a separate user process, a bug in one of these can crash that component, but cannot crash the entire system.
- A bug in the audio driver will cause the sound to be garbled or stop, but will not crash the computer.
- With the exception of OS X, which is based on the Mach microkernel, common desktop OS do not use microkernels.
- They are dominant in real-time, industrial, avionics, and military applications that are mission critical and have very high reliability requirements.

- The better-known microkernels include: **Integrity, K42, L4, PikeOS, QNX, Symbian, and MINIX 3**
- **MINIX 3 -** the idea of modularity to the limit, breaking most of the operating system up into a number of independent user-mode processes
- MINIX 3 is a POSIX-conformant, open source system.
- It is only about 12,000 lines of C and some 1400 lines of assembler for very low-level functions such as catching interrupts and switching processes.
- The device driver for the clock is in the kernel because the scheduler interacts closely with it.
- The other device drivers run as separate user processes.

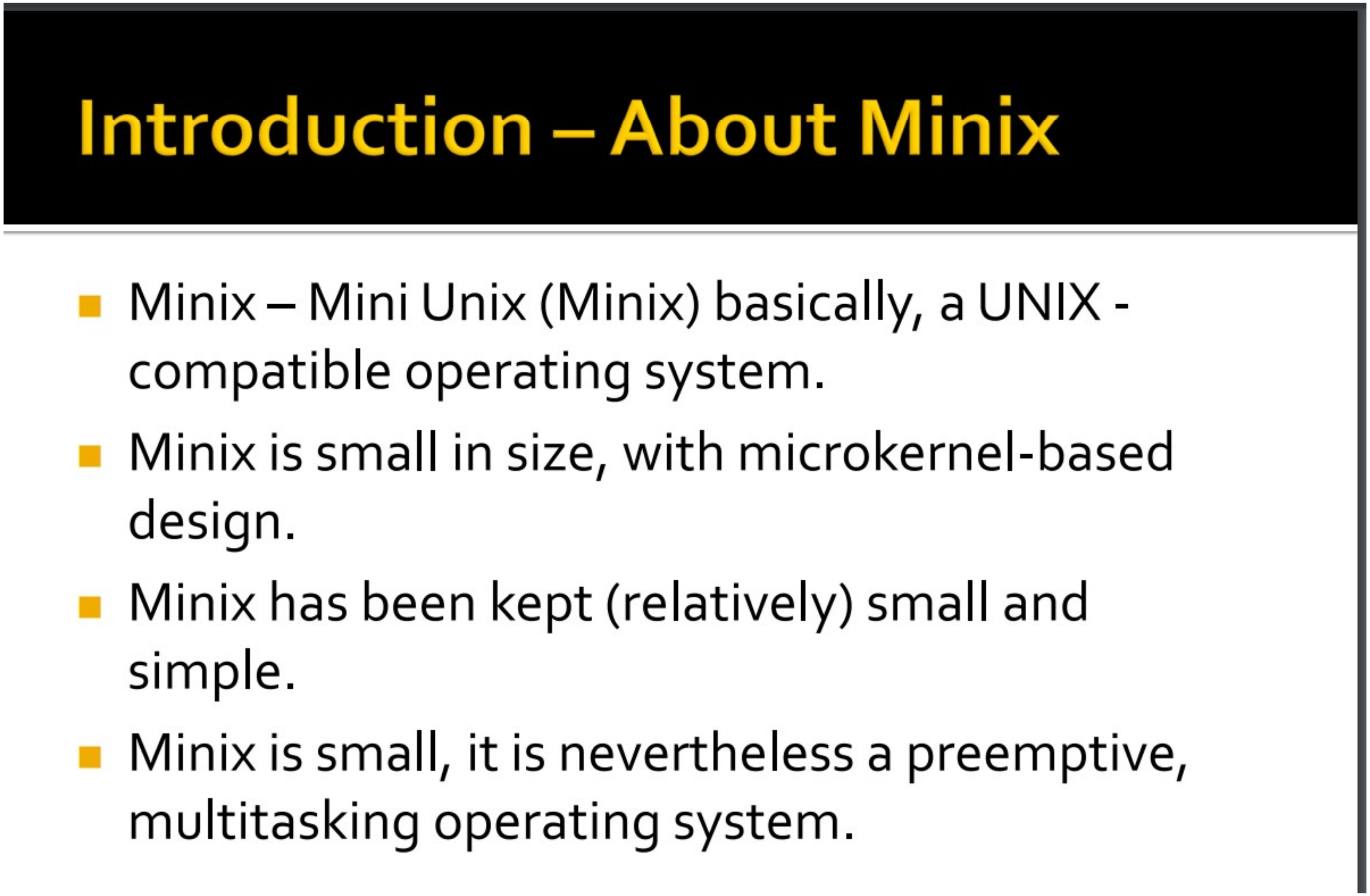
Simplified Structure of the MINIX System


Process
User programs
Servers
Drivers
User mode
Shell
Make
...
Other
FS
Proc.
Reinc.
...
Other
Disk
TTY
Netw
Print
...
Other
Microkernel handles interrupts, processes, scheduling, interprocess communication
Clock
Sys

- Outside the kernel, the system is structured as three layers of processes all running in user mode.
- The **lowest layer** - the device drivers.
- They run in user mode, they do not have physical access to the I/O port space and cannot issue I/O commands directly.
- **To program an I/O device**, the driver builds a structure telling which values to write to which I/O ports and makes a kernel call telling the kernel to do the write.
- This approach means that the kernel can check to see that the driver is writing (or reading) from I/O it is authorized to use.
- A buggy audio driver cannot accidentally write on the disk.

- One or more file servers manage the file system(s), the process manager creates, destroys, and manages processes, and so on.
- User programs obtain OS services by sending short messages to the servers asking for the POSIX system calls.
- **Eg:** a process needing to do a read sends a message to one of the file servers telling it what to read.
- **The reincarnation server** - to check if the other servers and drivers are functioning correctly
- In the event that a faulty one is detected, it is automatically replaced without any user intervention.
- The system is self-healing and can achieve high reliability

- Drivers can touch only authorized I/O ports; but access to kernel calls - controlled on a per-process basis, as is the ability to send messages to other processes.
- Processes can grant limited permission for other processes to have the kernel access their address spaces.
- A **simple scheduling algorithm** - to assign a numerical priority to every process and then have the kernel run the highest-priority process that is runnable.
- The policy—assigning priorities to processes—can be done by user-mode processes.
- Policy and mechanism can be decoupled and the kernel can be made smaller.



Introduction – About Minix

- Minix – Mini Unix (Minix) basically, a UNIX - compatible operating system.
- Minix is small in size, with microkernel-based design.
- Minix has been kept (relatively) small and simple.
- Minix is small, it is nevertheless a preemptive, multitasking operating system.

POSIX (Portable Operating System Interface) is a set of standard operating system interfaces based on the Unix operating system.

The need for Standardization arose because enterprises using computers wanted to be able to develop programs that could be moved among different manufacturer's computer systems without having to be recoded.

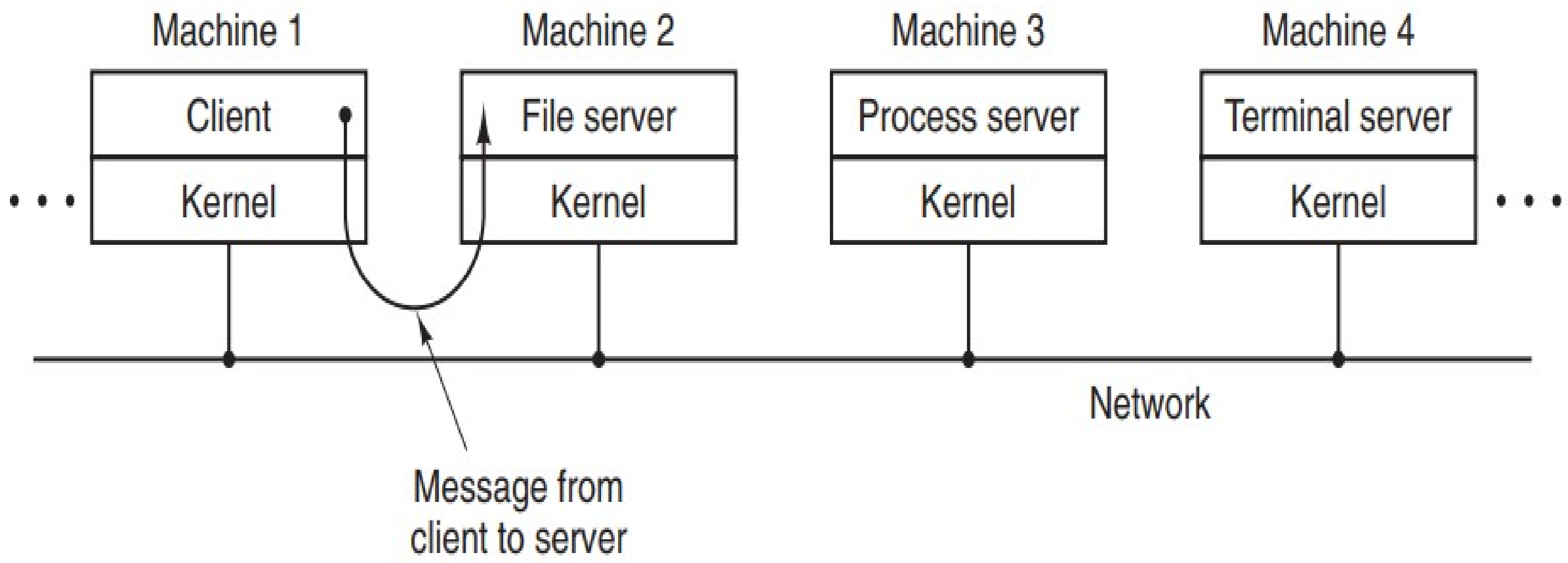
Unix was selected as the basis for a standard system interface partly because it was "manufacturer-neutral." However, several major versions of Unix existed so there was a need to develop a common denominator system.

Client-Server Model

- **Client-server model** - two classes of processes, the servers, each of which provides some service, and the clients, which use these services
- Lowest layer – **microkernel**
- The essence is the presence of client processes and server processes.
- Communication between clients and servers - **message passing**
- The client-server model is an abstraction that can be used for a single machine or for a network of machines.
- Many systems involve users at their home PCs as clients and large machines elsewhere running as servers; Web operates this way.

innovate achieve lead

The Client-Server Model over a Network


Machine 1
Machine 2
Machine 3
Machine 4
Client
File server
Process server
Terminal server
Kernel
Kernel
Kernel
Kernel
• • •
• • •
Network
Message from client to server

Virtual Machines

- The initial releases of **OS/360** were strictly batch systems
- The **official IBM timesharing system**, TSS/360, so big and slow that few sites converted to it
- **The z/VM** - widely used on IBM's current mainframes
- **The zSeries** - heavily used in large corporate data centers
- For example, as e-commerce servers that handle hundreds or thousands of transactions per second and use databases whose sizes run to millions of gigabytes.

VM/370

- Originally called CP/CMS Control Program/Cambridge Monitor System)and later renamed VM/370
- A timesharing system provides multiprogramming and an extended machine with a more convenient interface than the bare hardware.
- The essence of VM/370 is to completely separate these two functions.
- **Virtual machine monitor** - runs on the bare hardware and does the multiprogramming; but several virtual machines to the next layer up.
- They are exact copies of the bare hardware, including kernel/user mode, I/O, interrupts, and everything else the real machine has.

The Structure of VM/370 with CMS

- Different virtual machines can run different operating systems
- **Original IBM VM/370 system** – some ran OS/360 or one of the other large batch or transaction-processing operating systems; others ran a single-user, interactive system called CMS (Conversational Monitor System) for interactive timesharing users
- **CMS program** executed a system call - trapped to the operating system in its own virtual machine - run on a real machine instead of a virtual one
- Issue the normal hardware I/O instructions for reading its virtual disk or whatever was needed to carry out the call

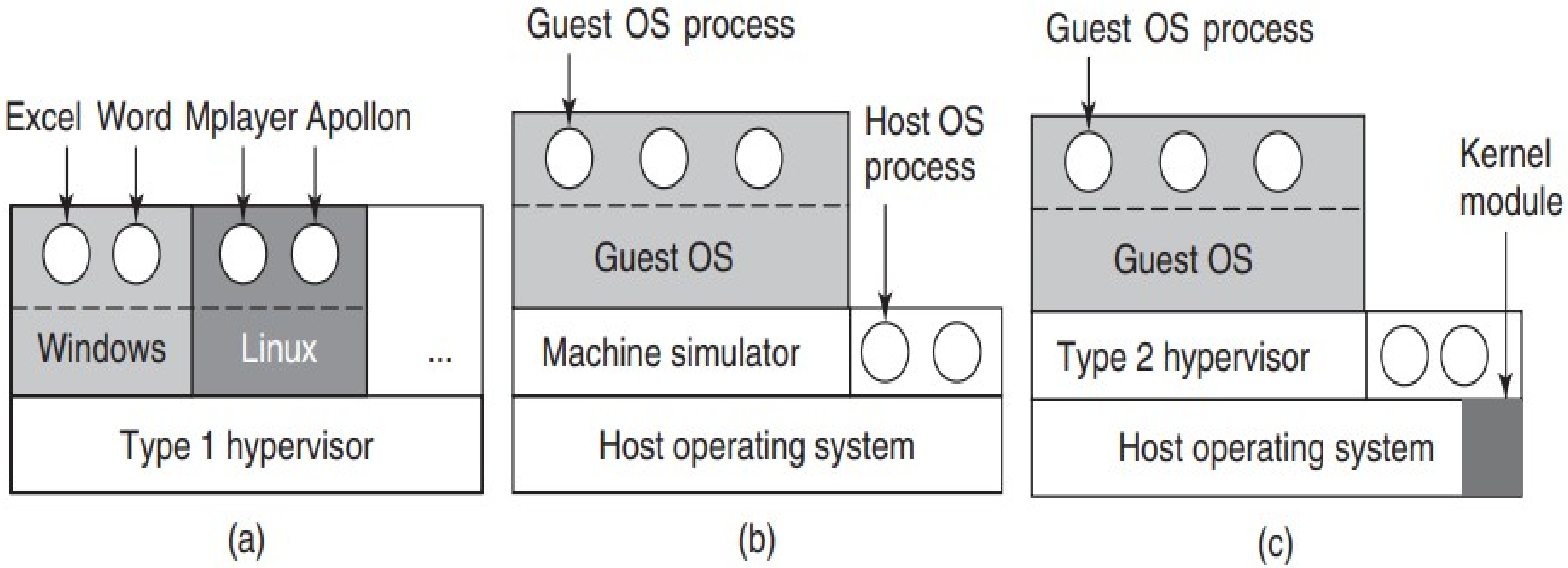
- **I/O instructions** were trapped by VM/370; then performed them as part of its simulation of the real hardware
- **z/VM** - used to run multiple complete operating systems rather than stripped-down single-user systems like CMS
- **For example**, the zSeries is capable of running one or more Linux virtual machines along with traditional IBM operating systems.

Virtual Machines Rediscovered

- **IBM, Oracle and Hewlett-Packard** - virtual-machine support to their high-end enterprise servers; virtualization has largely been ignored in the PC world until recently
- Many companies run their mail servers, Web servers, FTP servers, and other servers on separate computers, sometimes with different operating systems.
- **Virtualization**, a way to run them all on the same machine without having a crash of one server bring down the rest.

- **Virtualization** - popular in the Web hosting world
- Without virtualization - Web hosting customers are forced to choose between shared hosting and dedicated hosting
- End users who want to run two or more operating systems at the same time, say Windows and Linux, because some of their favourite application packages run on one and some run on the other.
- **Virtual machine monitor** - renamed type 1 hypervisor
- It requires more keystrokes than people are prepared to put up with now.

(a) A type 1 hypervisor (b) A pure type 2 hypervisor (c) A practical type 2 hypervisor


Excel
Word
Mplayer
Apollon
Windows
Linux
...
Type 1 hypervisor
(a)
Guest OS process
Host OS process
Guest OS
Machine simulator
Host operating system
(b)
Guest OS process
Kernel module
Guest OS
Type 2 hypervisor
Host operating system
(c)

- The research papers - commercial products (e.g., VMware Workstation and Xen) and a revival of interest in virtual machines.
- Besides VMware and Xen, popular hypervisors today include KVM (for the Linux kernel), VirtualBox (by Oracle), and Hyper-V (by Microsoft).
- **Binary translation** - improve the resulting systems; to add a kernel module to do some of the heavy lifting
- **Type 2 hypervisors** - all commercially available hypervisors, such as VMware Workstation, use this hybrid strategy

- **Type 2** makes uses of a host operating system and its file system to create processes, store files, and so on
- A type 1 hypervisor has no underlying support and must perform all these functions itself.
- After a type 2 hypervisor is started, it reads the installation CD-ROM (or CDROM image file) for the chosen guest operating system and installs the guest OS on a virtual disk, which is just a big file in the host operating system’s file system.
- Type 1 hypervisors cannot do this because there is no host operating system to store files on. They must manage their own storage on a raw disk partition.
- **Paravirtualization** - a different approach to handling control instructions is to modify the operating system to remove them

The Java Virtual Machine

- When Sun Microsystems invented the Java programming language, it also invented a virtual machine (i.e., a computer architecture) called the JVM (Java Virtual Machine).
- The Java compiler produces code, is executed by a software JVM interpreter
- **Advantage** - the JVM code can be shipped over the Internet to any computer that has a JVM interpreter and run there
- If the interpreter is implemented properly, which is not completely trivial, incoming JVM programs can be checked for safety and then executed in a protected environment so they cannot steal data or do any damage

Exokernels

- Giving each user a subset of the resources
- One virtual machine might get disk blocks 0 to 1023, the next one might get blocks 1024 to 2047, and so on.
- **Exokernel** - a program running in kernel mode at the bottom layer
- Allocate resources to virtual machines and then check attempts to use them to make sure no machine is trying to use somebody else's resources
- Each user-level virtual machine can run its own OS, as on VM/370 and the Pentium virtual 8086s, except that each one is restricted to using only the resources it has asked for and been allocated.

Advantage of the Exokernel Scheme

- It saves a layer of mapping
- In the other designs, each virtual machine thinks it has its own disk, with blocks running from 0 to some maximum, so the virtual machine monitor must maintain tables to remap disk addresses (and all other resources).
- With the exokernel, this remapping is not needed.
- The exokernel need only keep track of which virtual machine has been assigned which resource.
- Separating the multiprogramming (in the exokernel) from the user operating system code (in user space) but with less overhead

THANK YOU

BITS Pilani
Pilani Campus

BITS Pilani
Pilani Campus

COMPUTING AND DESIGN
SESSION 16

Course design by - Dr. Lucy J. Gudino &
Dr Chandrasekhar
Faculty – Thangakumar J

BITS Pilani
Pilani Campus

Operating Systems – Abstraction vs. Realization (Cont..)

Last Session

List of Topic Title	Text/Ref Book/external resource
Operating System Structure • Monolithic Systems • Layered Systems • Microkernels • Client Server Model • Virtual Machines	R1 (1.7)

OS: Abstraction vs. Realization (Cont..)

Today's topic

- Principle of Virtualization
- Benefits and features
- Building Blocks
- Types of Virtual Machines and Implementations
- Virtualization and Operating System Components

Recap of concepts

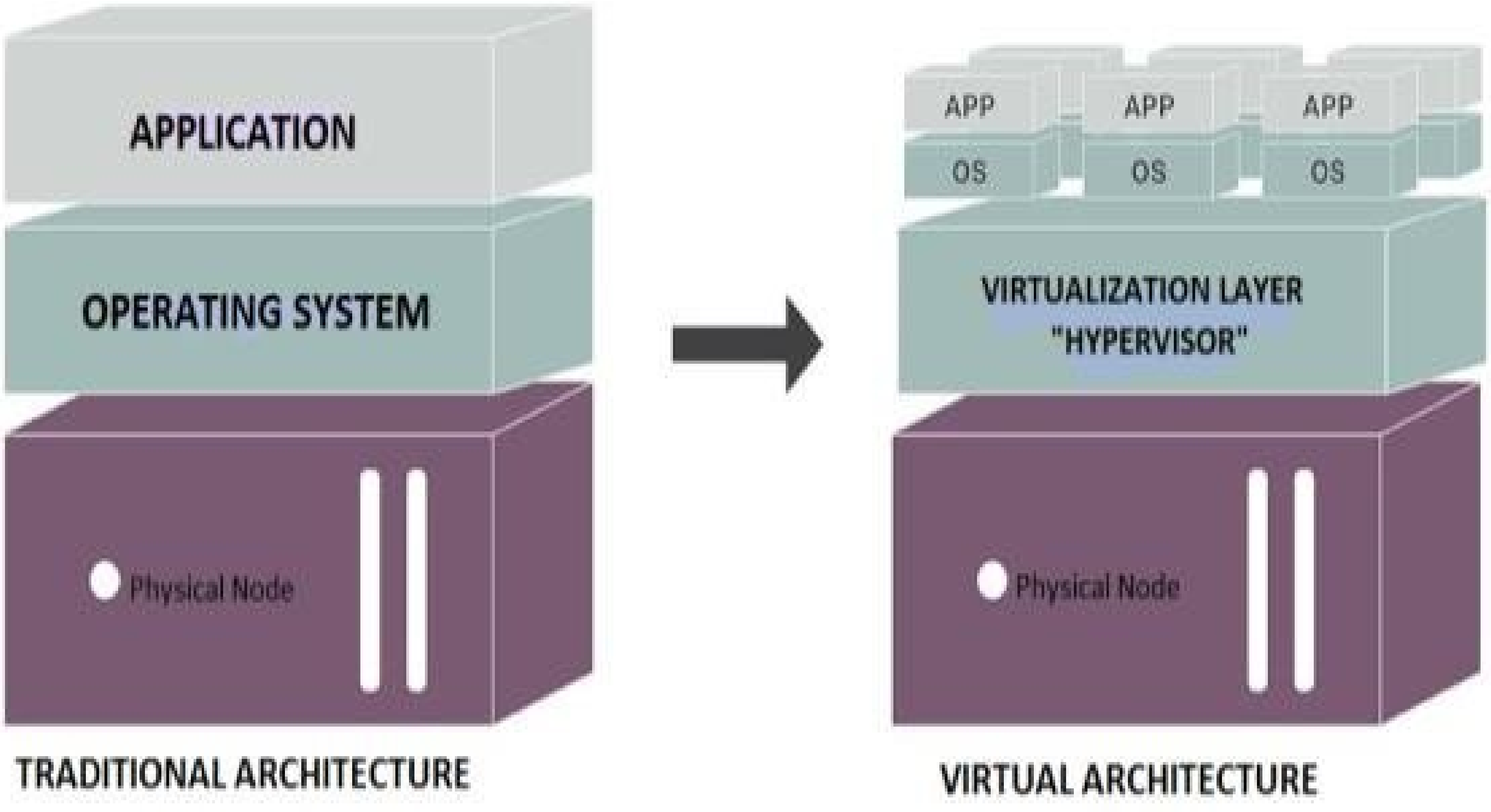
A virtual machine is **a computer file, typically called an image, that behaves like an actual computer**. It can run in a window as a separate computing environment, often to run a different operating system—or even to function as the user's entire computer experience—as is common on many people's work computers.

Example

A process virtual machine **allows you to run a single process as an application on a host machine**. An example of a process virtual machine is the Java Virtual Machine (JVM) which allows any system to run Java applications as if they were native to the system.

VIRTUALIZATION

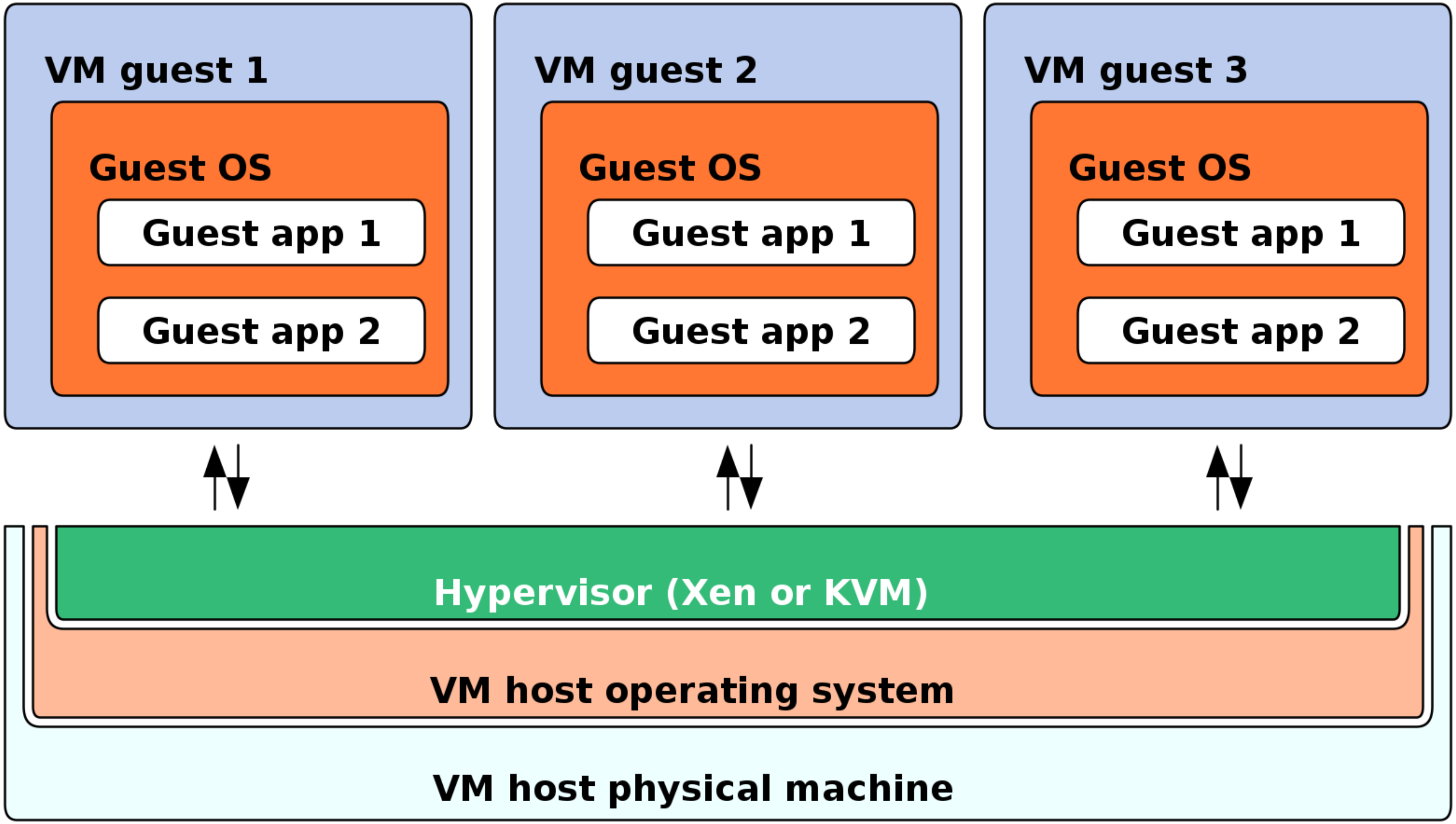
- The provision of a set of computing resources or their logical combination, abstracted from the hardware while providing logical isolation of computing processes running on the same physical resource.
- This includes operating systems, hardware platforms (computers and servers), storage arrays (file storage), and computer network resources.
- Virtualization is realized with the help of software – a **hypervisor**.
- It manages the physical resources of a computing machine and distributes those resources among several different operating systems, allowing them to run simultaneously.


APPLICATION
OPERATING SYSTEM
Physical Node
TRADITIONAL ARCHITECTURE
APP
APP
APP
OS
OS
OS
VIRTUALIZATION LAYER
"HYPERVISOR"
Physical Node
VIRTUAL ARCHITECTURE

- The hypervisor creates multiple copies, clones of its hardware resources, out of a single physical computer.
- A guest operating system can be installed on each virtual machine, not tied to the host hardware.
- A hypervisor allows **virtualizing desktops, applications, storage systems, and networks.**
- It can be either installed directly on the hardware or operate as a layer installed in the operating system between the hardware and the virtual machine.
  - **A virtualization host** is a physical server with a hypervisor running on it.
  - **A host system** is the physical machine OS that has the installed hypervisor.
  - **A guest operating system** is an OS running inside the virtual machine.
  - **A virtual machine** is a virtual copy of real hardware, an application that functions as a physical computer or server.

PRINCIPLE OF VIRTUALIZATION

- A technique of how to separate a service from the underlying physical delivery of that service.
- The process of creating a virtual version of something like computer hardware.
- Multiple OS and applications can run on same machine and its same hardware at the same time, increasing the utilization and flexibility of hardware.
- It allows to share a single physical instance of a resource or an application among multiple customers and organizations at one time.



VM guest 1

Guest OS

Guest app 1

Guest app 2

VM guest 2

Guest OS

Guest app 1

Guest app 2

VM guest 3

Guest OS

Guest app 1

Guest app 2

Hypervisor (Xen or KVM)

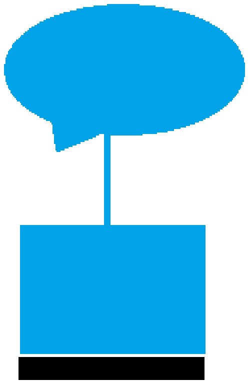
VM host operating system

VM host physical machine

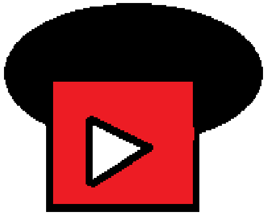
Types of Virtualization

- **Application Virtualization:** helps a user to have remote access of an application from a server. The server stores all personal information and other characteristics of the application but can still run on a local workstation through the internet.
- **Network Virtualization:** provides a facility to create and provision virtual networks—logical switches, routers, firewalls, load balancer, Virtual Private Network (VPN), and workload security within days or even in weeks.
- **Desktop Virtualization:** allows the users' OS to be remotely stored on a server in the data centre. The user to access their desktop virtually, from any location by a different machine.

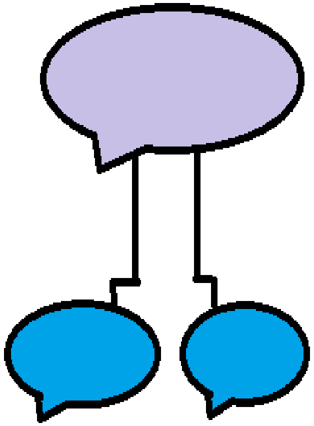
TYPES OF VIRTUALIZATION



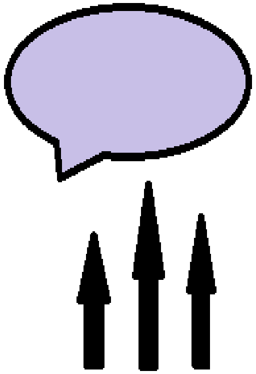
DESKTOP
VIRTUALIZATION



APPLICATION
VIRTUALIZATION



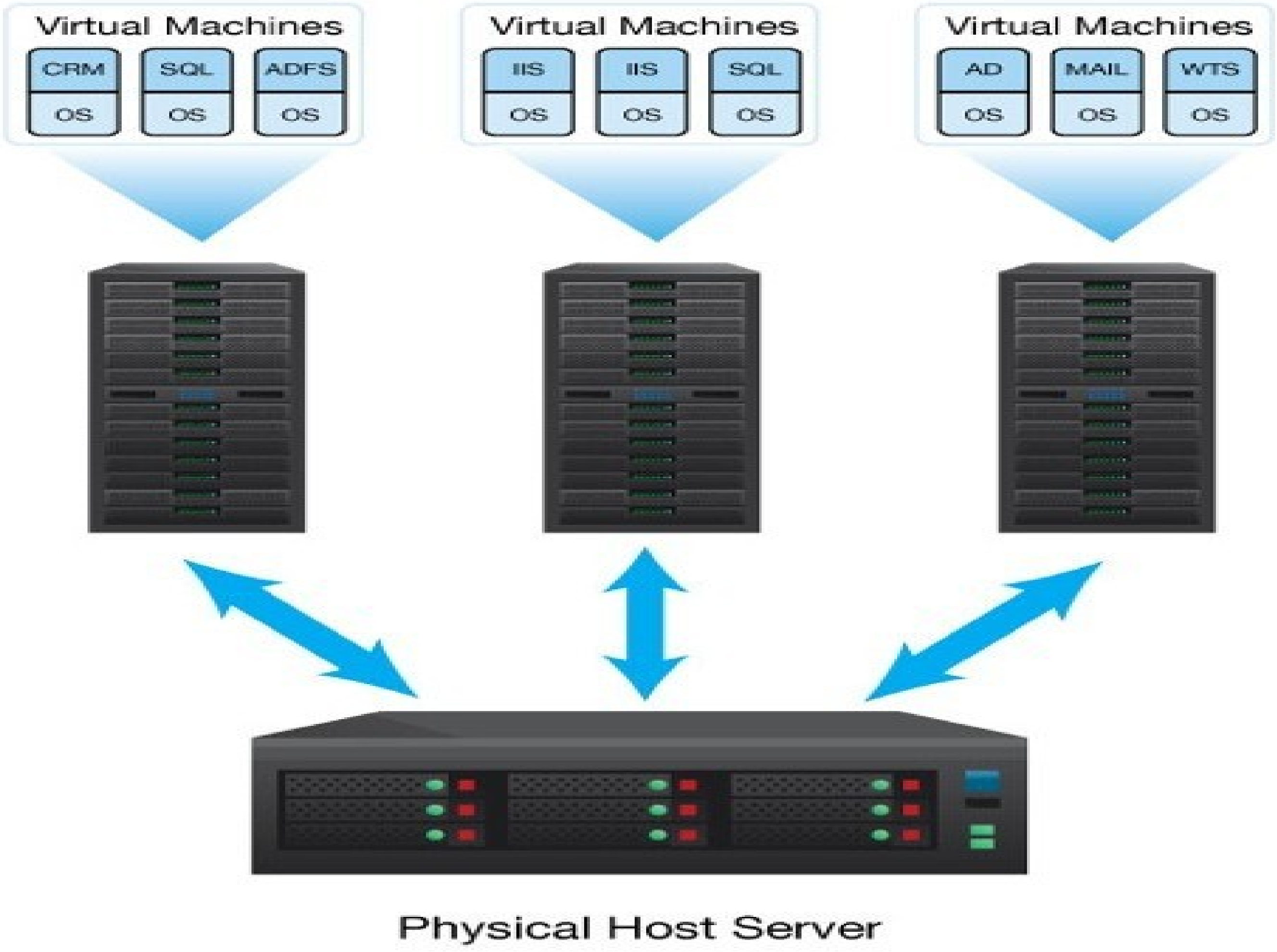
NETWORK
VIRTUALIZATION



STORAGE
VIRTUALIZATION

- **Storage Virtualization:** an array of servers that are managed by a virtual storage system
- **Server Virtualization:** the central-server(physical server) is divided into multiple different virtual servers by changing the identity number, processors. So, each system can operate its own operating systems in isolate manner.
- **Data virtualization:** the data is collected from various sources and managed that at a single place without knowing more about the technical information. big giant companies are providing their services like Oracle, IBM, At scale, Cdata, etc.

Server Virtual Machine


Virtual Machines
CRM
SQL
ADFS
OS
OS
OS
Virtual Machines
IIS
IIS
SQL
OS
OS
OS
Virtual Machines
AD
MAIL
WTS
OS
OS
OS
Physical Host Server

BENEFITS OF VIRTUALIZATION

1. **More flexible and efficient allocation of resources.**
2. **Enhance development productivity.**
3. **It lowers the cost of IT infrastructure.**
4. **Remote access and rapid scalability.**
5. **High availability and disaster recovery.**
6. **Pay peruse of the IT infrastructure on demand.**
7. **Enables running multiple operating systems.**

Virtual Machines

- It is a fake computer system operating on hardware.
- It partially uses the hardware of your system (like CPU, RAM, disk space, etc.) but its space is completely separated from main system.
- **Virtual Machine Language:** can be understood by different operating systems; platform-independent
- To use code that can be executed on different types of operating systems like (Windows, Linux, etc), the virtual machine language will be helpful.

Virtual Machine Components

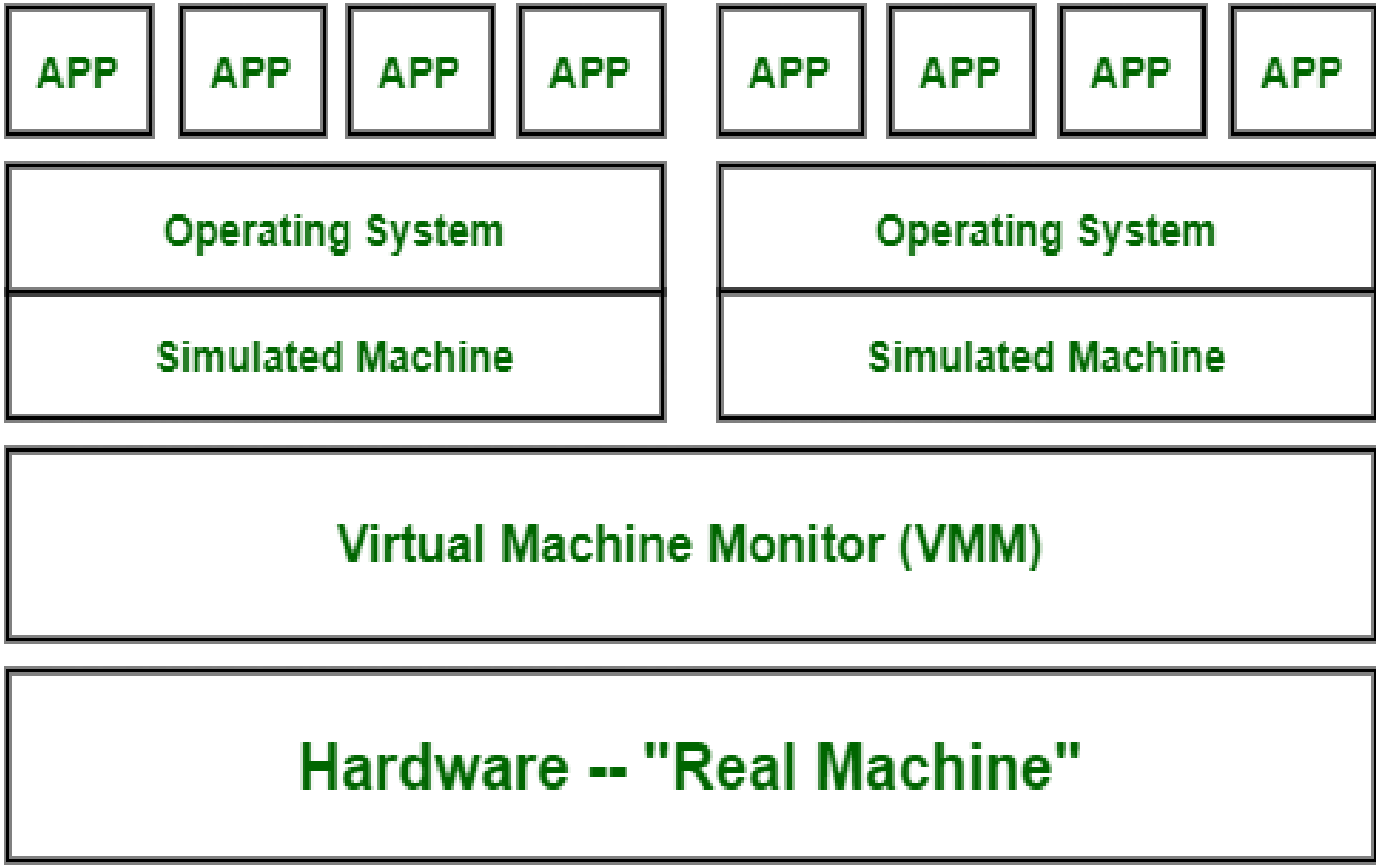
- **Operating System**
- **VMware Tools**
- **Compatibility Setting**
- **Hardware Devices**

TYPES OF VIRTUAL MACHINES AND IMPLEMENTATIONS

1. System Virtual Machine

- It gives us complete system platform and gives the execution of the complete virtual operating system.
- Provides an environment for an OS to be installed completely.

System Virtual Machine



APP APP APP APP

APP APP APP APP

Operating System

Simulated Machine

Operating System

Simulated Machine

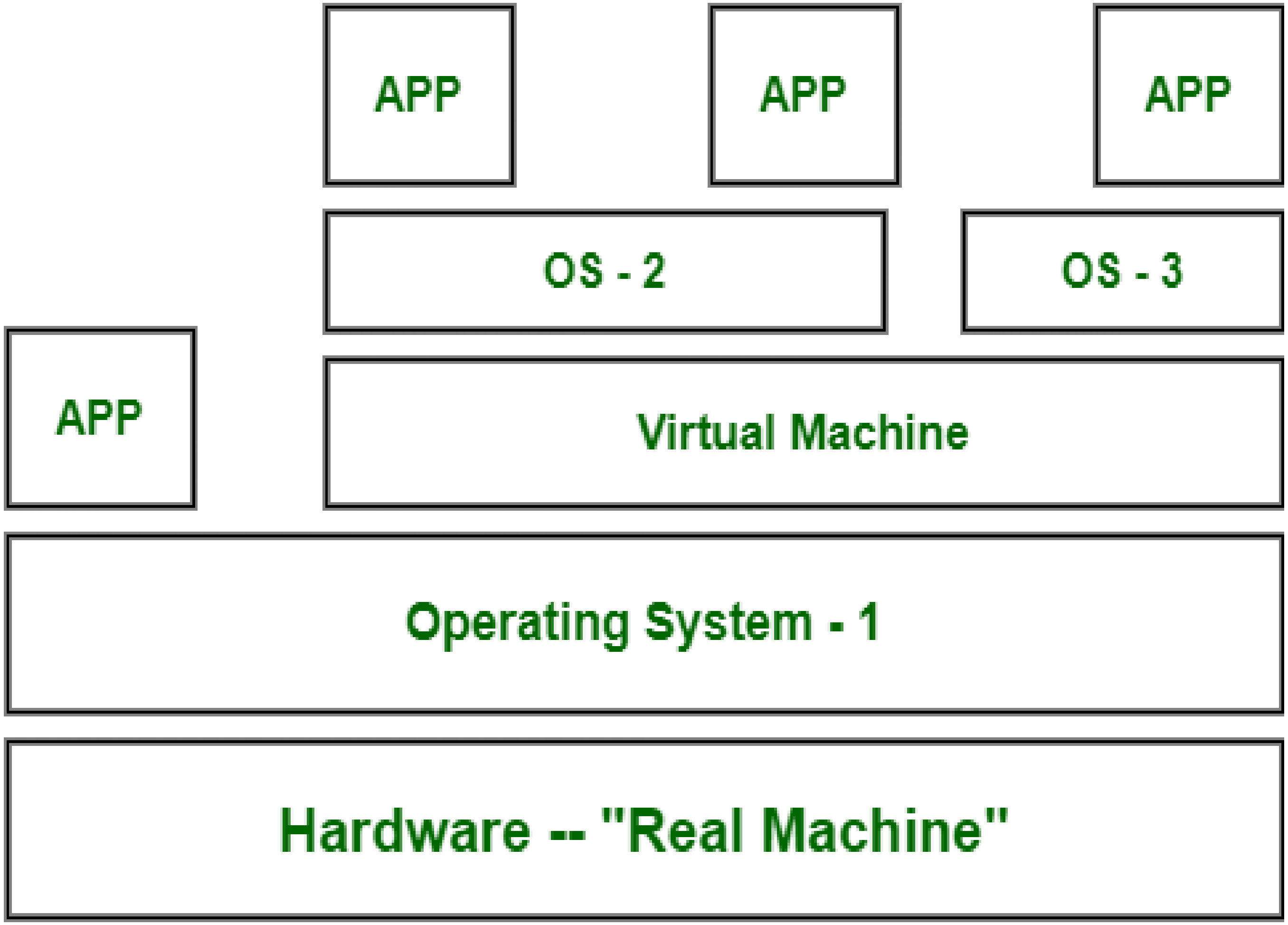
Virtual Machine Monitor (VMM)

Hardware -- "Real Machine"

2. Process Virtual Machine

- It does not provide with the facility to install the virtual operating system completely.
- It creates virtual environment of OS while using some app or program and this environment will be destroyed as soon as we exit from that app.

Process Virtual Machine


APP
APP
APP
OS - 2
OS - 3
APP
Virtual Machine
Operating System - 1
Hardware -- "Real Machine"

How are Virtual Machines Implemented?

- Virtual Machines are written in portable and efficient programming language (C or C++).
- For performance critical components, assembly language is used.
- Some virtual machines (Lisp, Forth, Smalltalk) are largely written in language itself.
- Most Java VM implementations consists of a mixture of C/C++ and assembly code.

Advantages - Virtual Machines

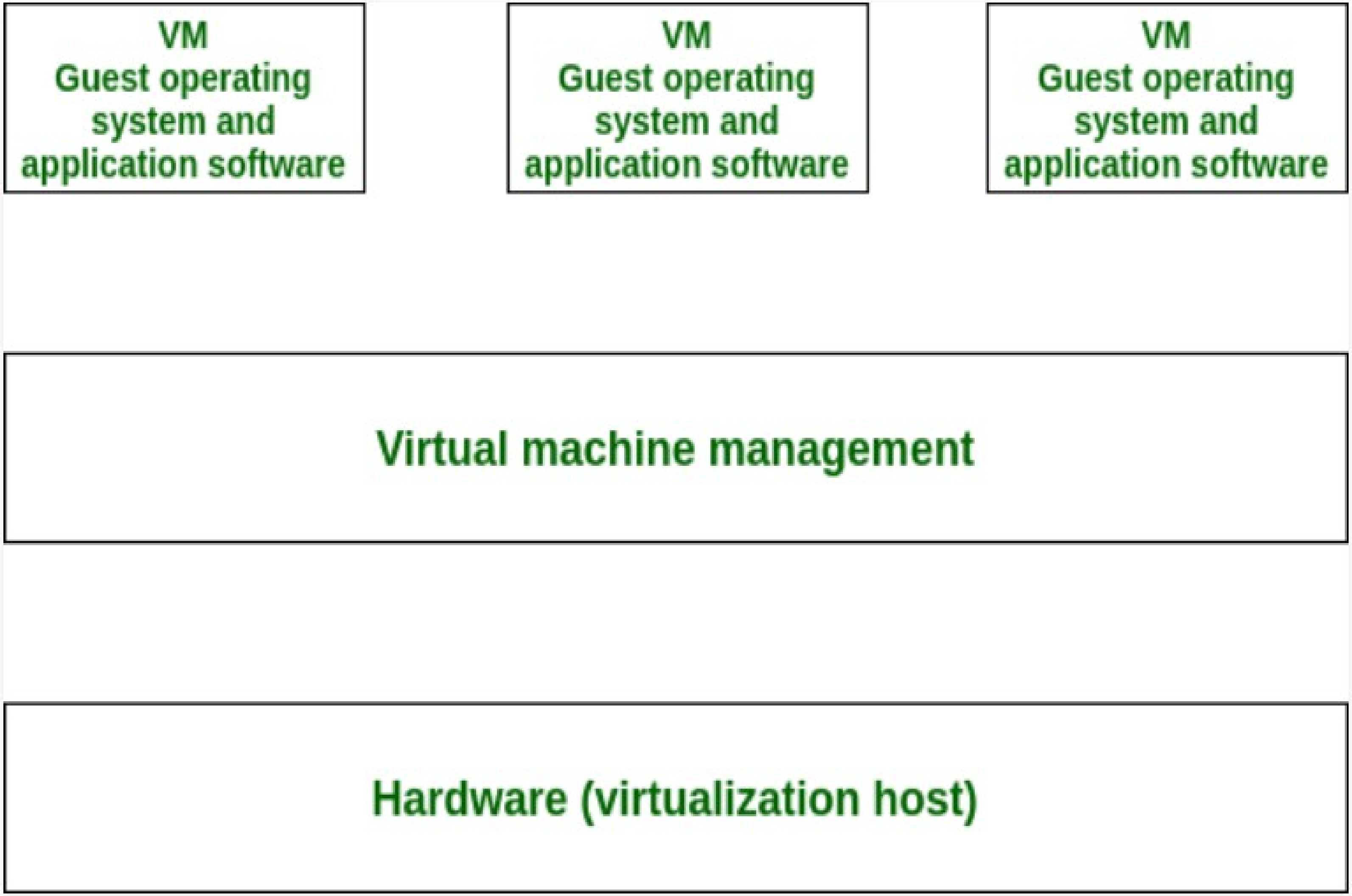
- Provide software compatibility to the software running on it
- Provides isolation between different types of operating systems and processes
- Provide encapsulation and software present on a virtual machine can be modified and controlled
- Provide various features like no dual booting, transfer of files between virtual machines, error in one OS doesn’t affect the other OS present on the host, freash OS can be added easily.
- Provide good software management like these can run a complete software stack of the host machine, run legacy OS, etc..

VIRTUALIZATION AND OPERATING SYSTEM COMPONENTS

- It refers to an **operating system feature in which the kernel enables the existence of various isolated user-space instances**.
- The installation of virtualization software refers to Operating system-based virtualization.
- It is installed over a pre-existing operating system and that operating system is called the host operating system.
- Virtualization software converts hardware IT resources that require unique software for operation into virtualized IT resources.

How Operating System Virtualization Works?

- The OS manages all the software and hardware of the computer.
- Several different computer programs can run at the same time.
- This is done by using the CPU of the computer.
- With the combination of few components of the computer which is coordinated by the operating system, every program runs successfully.



VM
Guest operating system and application software

VM
Guest operating system and application software

VM
Guest operating system and application software

Virtual machine management

Hardware (virtualization host)

- Operating system-based services are: **Backup and Recovery, Security Management, Integration to Directory Services.**
- Various major operations of Operating System Based Virtualization are:

1. *Hardware capabilities can be employed, such as the network connection and CPU.*
2. *Connected peripherals with which it can interact, such as webcam, printer, keyboard, or Scanners.*
3. *Data that can be read or written, such as files, folders, and network shares.*

- Operating system-based virtualization can raise demands and problems related to performance overhead, such as:

1. *The host operating system employs CPU, memory, and other hardware IT resources.*
2. *Hardware-related calls from guest operating systems need to navigate numerous layers to and from the hardware, which shrinkage overall performance.*
3. *Licenses are frequently essential for host operating systems, in addition to individual licenses for each of their guest operating systems.*

Types of OS Virtualization

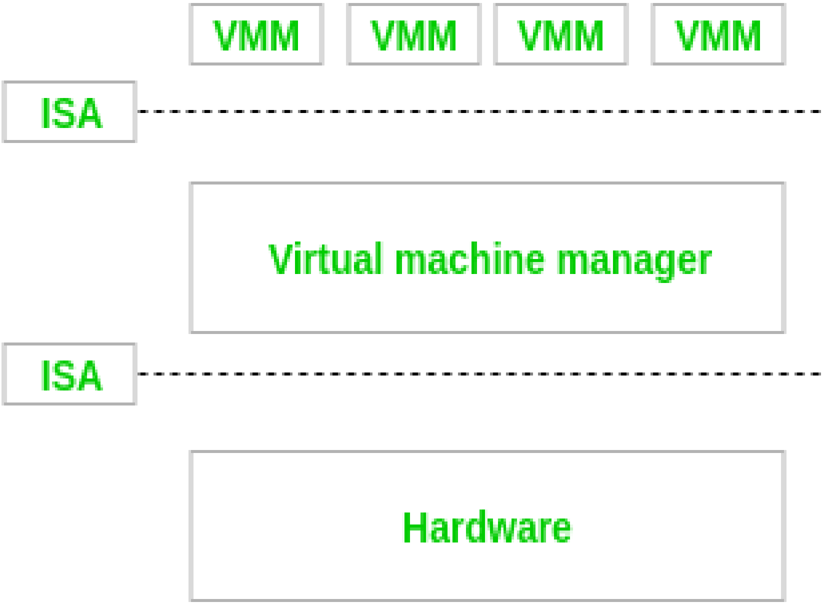
- **Linux Operating System virtualization:** VMware Workstation software is used to virtualize Linux systems.
- In addition, to install any software by the means of virtualization the user will need VMware software to install first.
- **Windows Operating System virtualizations:** This type of virtualization is similar to the above to install any software there is a need to install VMware software firstly.

Advantages of OS Virtualization

- Eliminates the use of physical space which utilizes by the IT system
- No hardware required the maintenance will be less; save both time and money.
- **Lower power consumption, lower cooling requirement, low maintenance, and more electricity savings.**
- It allows the companies to make enhancement in terms of efficiency to use the server hardware and thus there is a greater return on investment (ROI) on the purchase and greater operational works.
- **Quick deployment capability**

Hardware Based Virtualization

- **Hardware based virtualization** - A platform virtualization approach that allows efficient full virtualization with the help of hardware capabilities, primarily from the host processor.
- **Hardware-level virtualization** - An abstract execution environment in terms of computer hardware in which guest OS can be run
- A fundamental component of hardware virtualization is the **hypervisor, or virtual machine manager (VMM)**.
- Two types of Hypervisors:
    - **Type-I hypervisors**
    - **Type-II hypervisors**



VMM VMM VMM VMM

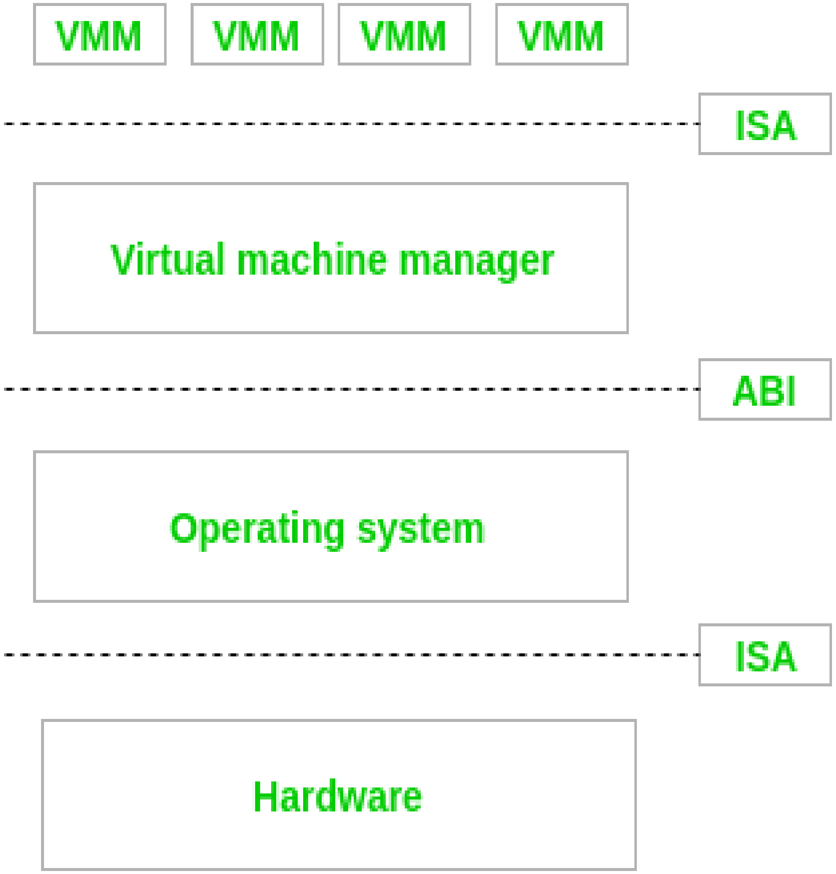
ISA

Virtual machine manager

ISA

Hardware

Type-I Hypervisor



VMM VMM VMM VMM

ISA

Virtual machine manager

ABI

Operating system

ISA

Hardware

Type-II Hypervisor

Advantages of Hardware-based Virtualization

- Reduces the maintenance overhead of paravirtualization
- Convenient to attain enhanced performance
- More Efficient Resource Utilization
- Lower Overall Costs Because Of Server Consolidation
- Increased Uptime Because Of Advanced Hardware Virtualization Features
- Increased IT Flexibility

Disadvantages of Hardware-based Virtualization

- Hardware-based virtualization requires explicit support in the host CPU, which may not available on all x86/x86_64 processors.
- A "pure" hardware-based virtualization approach, including the entire unmodified guest operating system, involves many VM traps, and thus a rapid increase in CPU overhead occurs which limits the scalability and efficiency of server consolidation.
- This performance hit can be mitigated by the use of para-virtualized drivers; the combination has been called "**hybrid virtualization**".

Quick overview of cloud

- Part 1: What is Cloud Computing?
- Part 2: Six Advantages of Cloud Computing
- Part 3: What is Amazon Web Services (AWS)?
- Part 4: The AWS Cloud Adoption Framework (CAF)

What is Cloud Computing?

Cloud computing is the **on-demand** delivery of compute power, database storage, applications, and other IT resources through a cloud services platform **via the internet** with **pay-as-you-go** pricing.

1001011010001
0100101010111100
1001110101
10010100101010

© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Before Cloud Computing

Cloud computing enables you to **stop thinking of your infrastructure as hardware**, and instead **think of it (and use it) as software.**

© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Before Cloud Computing

- Hardware solutions are **physical**. This means they require:
  - Space
  - Staff
  - Physical security
  - Planning
  - Capital expenditure
- Guess at theoretical maximum peaks
  - Is there enough resource capacity?
  - Do we have sufficient storage?

What if your needs change?

You have to go through the **time, effort, and cost** required to change all these.

© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Utilizing Cloud Computing

Software is flexible.

If your needs change, your software can change much more **quickly, easily, and cost-effectively** than your hardware.

© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Three Models of Cloud Computing

IaaS

Infrastructure
as a Service

PaaS

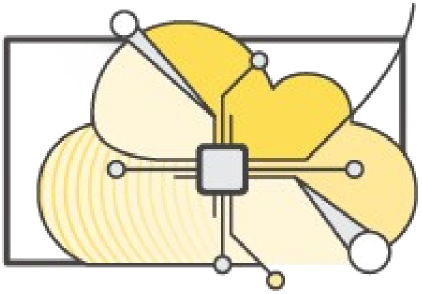
Platform
as a Service

SaaS

Software
as a Service

© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Three Cloud Deployment Models



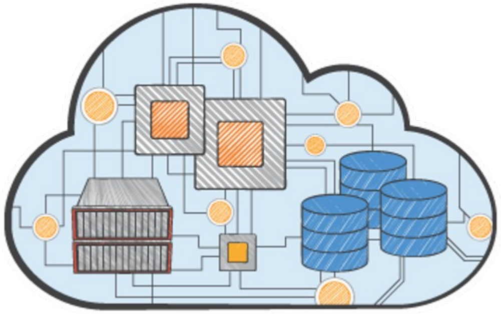
All-In Cloud

Hybrid

Private Cloud (On-premises)

© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

All-In Cloud versus On-Premises



All-In Cloud

- No upfront investment
- Low ongoing costs
- Focus on innovation
- Flexible capacity
- Speed and agility
- Global reach on demand

On-Premises

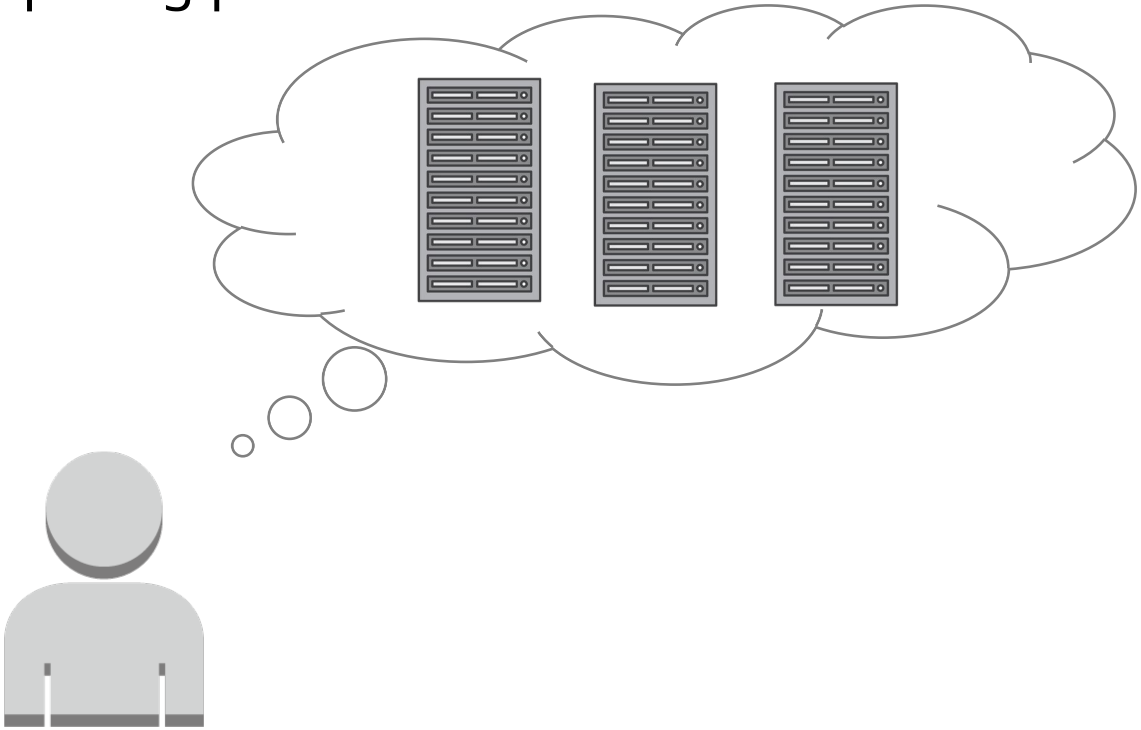
- Large initial purchase
- Labor, patches, and upgrade cycles
- Systems administration
- Fixed capacity
- Long procurement cycle and setup
- Limited geographic regions

© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

What can you do in the cloud?

You can use a cloud computing platform for:

- Application Hosting
- Backup and Storage
- Content Delivery
- Websites
- Enterprise IT
- Databases



© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Important Cloud Terminology

- **High Availability (Highly Available):**
  - Accessible when you need it
- **Fault Tolerance (Fault Tolerant):**
  - Ability to withstand a certain amount of failure and still remain functional
- **Scalability (Scalable):**
  - Ability to easily grow in size, capacity, and/or scope when required
  - Growth is (usually) based on demand
- **Elasticity (Elastic):**
  - Ability to grow (scale) when required and to reduce in size when resources are no longer needed

© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Trade **capital expense** for **variable expense**.

Benefit from **massive economies of scale**.

Eliminate guessing on your capacity needs.

Increase **speed** and **agility**.

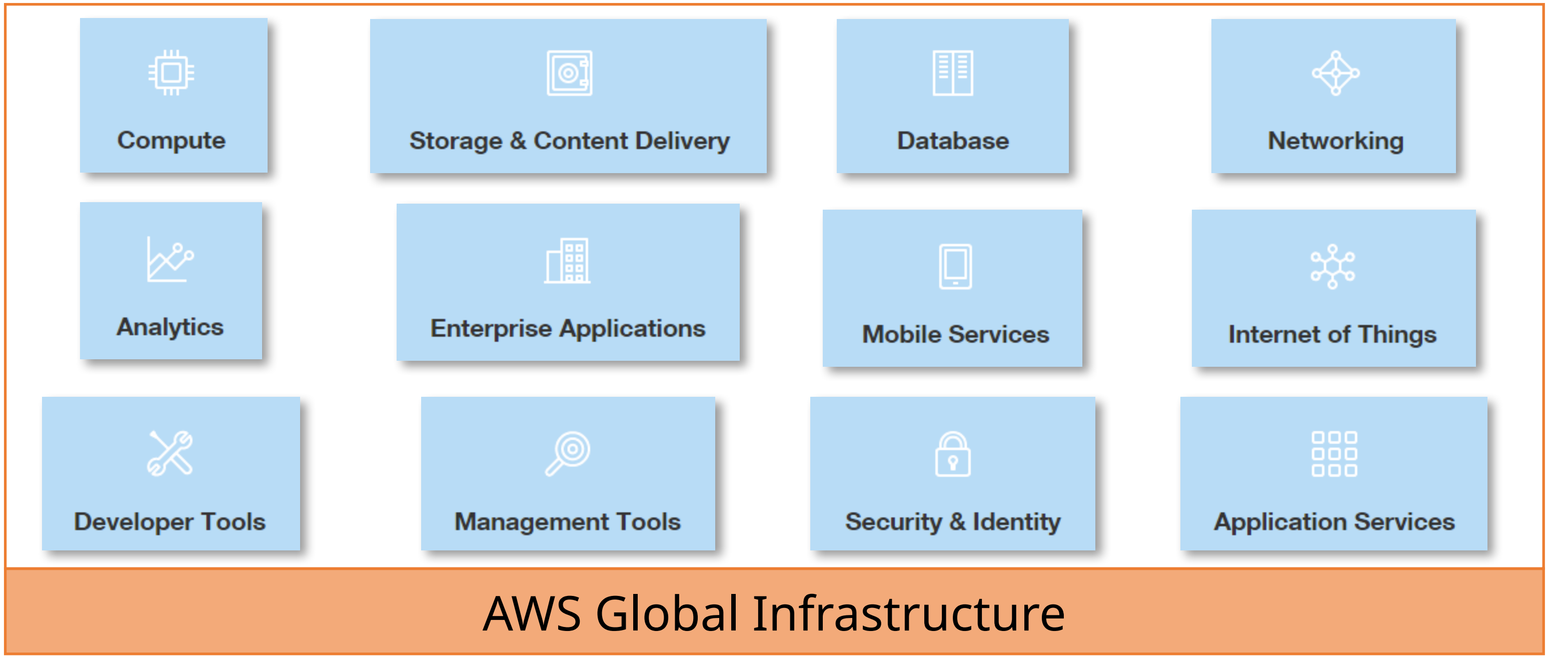
Stop spending money to run and maintain data centers.

Go global in minutes.

© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

What is AWS?

AWS is a secure cloud platform with more than 165 different services that include solutions for:



Compute

Storage & Content Delivery

Database

Networking

Analytics

Enterprise Applications

Mobile Services

Internet of Things

Developer Tools

Management Tools

Security & Identity

Application Services

AWS Global Infrastructure

© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

AWS by Category: Core Services

Compute

- Amazon EC2
- AWS Lambda
- Auto Scaling
- AWS Elastic Beanstalk
- Amazon Elastic Container Registry
- Amazon Elastic Container Service
- Amazon Lightsail
- AWS Batch

Networking

- Amazon VPC
- Amazon Route 53
- AWS Direct Connect
- Elastic Load Balancing

Storage

- Amazon S3
- Amazon EBS
- Amazon CloudFront
- Amazon Glacier
- Amazon Elastic File System
- AWS Snowball
- Storage Gateway
- AWS Snowmobile

Database

- Amazon RDS
- Amazon DynamoDB
- Amazon Redshift
- AWS Database Migration Service
- Amazon ElastiCache

© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Analytics

Amazon EMR

AWS Data Pipeline

Amazon Elasticsearch

Amazon Kinesis

Amazon Machine Learning

Amazon QuickSight

Amazon Redshift

Amazon Athena

Enterprise Apps

Amazon WorkSpaces

Amazon WorkMail

Amazon WorkDocs

Mobile Services

AWS Mobile Hub

Amazon SNS

Amazon Cognito

AWS Device Farm

Amazon Mobile Analytics

AWS Mobile SDKs

Amazon Pinpoint

Internet of Things

AWS IoT

AWS Greengrass

© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

AWS by Category:

Developer Tools

- AWS CodeCommit
- AWS CodeDeploy
- AWS CodePipeline
- AWS CodeBuild
- AWS X-Ray

Management Tools

- Amazon CloudWatch
- AWS CloudFormation
- AWS CloudTrail
- AWS Config
- AWS OpsWorks
- AWS Service Catalog
- AWS Trusted Advisor
- AWS Organizations

Security & Identity

- AWS Identity and Access Management
- AWS Directory Service
- Amazon Inspector
- AWS CloudHSM
- AWS Key Management Service
- AWS WAF
- AWS Certificate Manager
- AWS Shield

App Services

- Amazon API Gateway
- Amazon AppStream
- Amazon CloudSearch
- Amazon Elastic Transcoder
- Amazon SES
- Amazon SNS
- Amazon SQS
- Amazon SWF

© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Access to AWS Services

- AWS Management Console
  - Access on the go with AWS Console Mobile App
- AWS Command Line Interface (AWS CLI)
- Software Development Kits (SDK)

© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Motivation

Powerful multi-core processors

Explosion of domain applications

General purpose graphic processors

Proliferation of devices

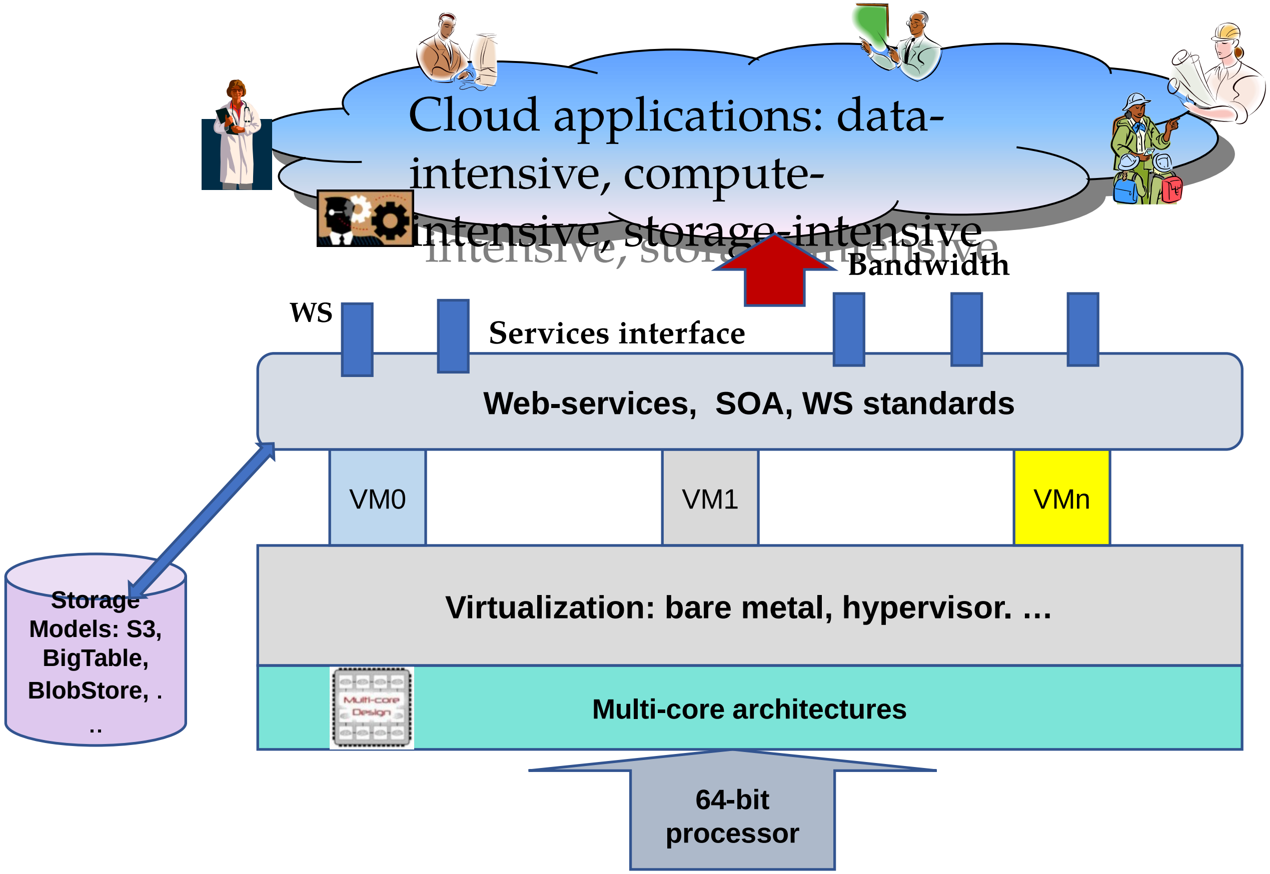
Superior software methodologies

Wider bandwidth for communication

Virtualization leveraging the powerful hardware

1. **Web Scale Problems**
2. **Web 2.0 and Social Networking**
3. **Information Explosion**
4. **Mobile Web**

Technology Advances



Cloud applications: data-intensive, compute-intensive, storage-intensive
Bandwidth
WS
Services interface
Web-services, SOA, WS standards
VM0
VM1
VMn
Storage Models: S3, BigTable, BlobStore, . ..
Virtualization: bare metal, hypervisor. …
Multi-core Design
Multi-core architectures
64-bit processor

What is Cloud Computing?

Cloud Computing is a general term used to describe a new class of network based computing that takes place over the Internet,

- basically a step on from Utility Computing
- a collection/group of integrated and networked hardware, software and Internet infrastructure (called a platform).
- Using the Internet for communication and transport provides hardware, software and networking services to clients

These platforms hide the complexity and details of the underlying infrastructure from users and applications by providing very simple graphical interface or API (Applications Programming Interface).

What is Cloud Computing cont.…

In addition, the platform provides on demand services, that are always on, anywhere, anytime and any place.

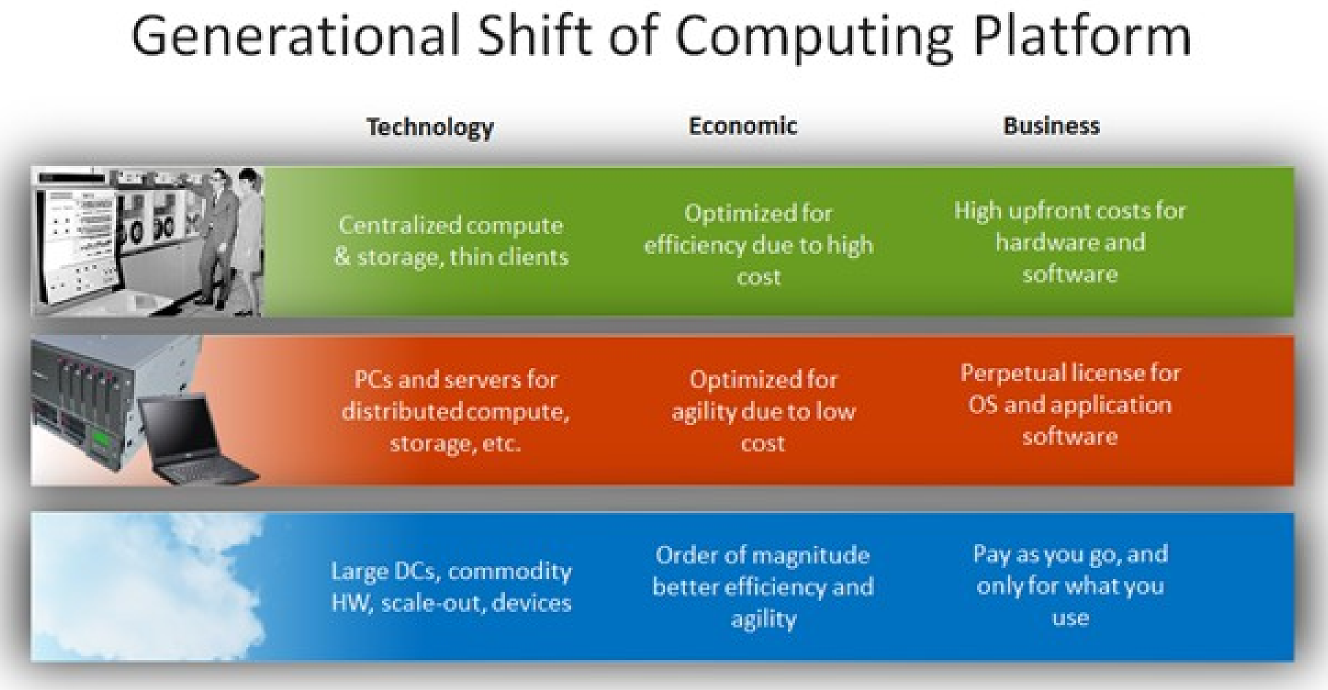
Pay for use and as needed, elastic

- scale up and down in capacity and functionalities

The hardware and software services are available to

- general public, enterprises, corporations and businesses markets

Drivers for the new Platform

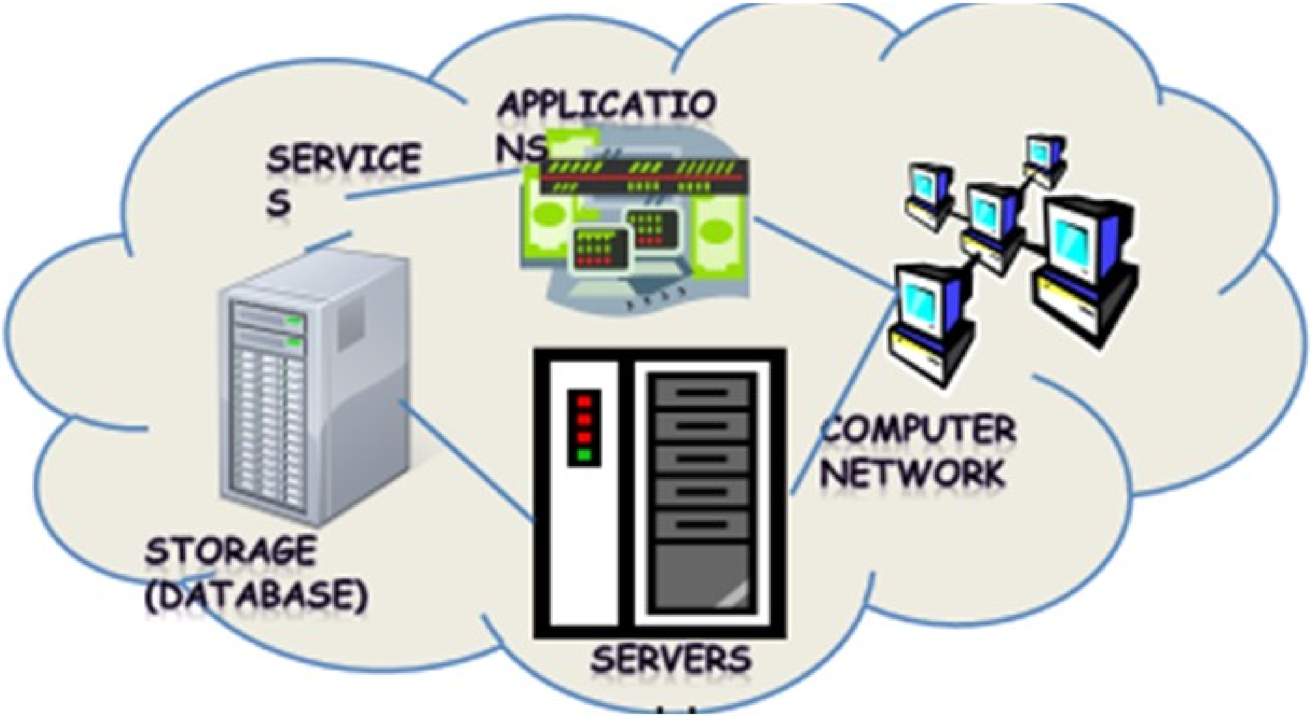


Generational Shift of Computing Platform

	Technology	Economic	Business
	Centralized compute & storage, thin clients	Optimized for efficiency due to high cost	High upfront costs for hardware and software
	PCs and servers for distributed compute, storage, etc.	Optimized for agility due to low cost	Perpetual license for OS and application software
	Large DCs, commodity HW, scale-out, devices	Order of magnitude better efficiency and agility	Pay as you go, and only for what you use

http://blogs.technet.com/b/yungchou/archive/2011/03/03/chou-s-theories-of-cloud-computing-the-5-3-2-principle.aspx

Cloud Summary


APPLICATIO NS
SERVICE S
COMPUTER NETWORK
STORAGE (DATABASE)
SERVERS

- Shared pool of configurable computing resources
- On-demand network access
- Provisioned by the Service Provider

Cloud Computing: Definition

The US National Institute of Standards (NIST) defines cloud computing as follows:
Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

3-4-5 rule of Cloud Computing

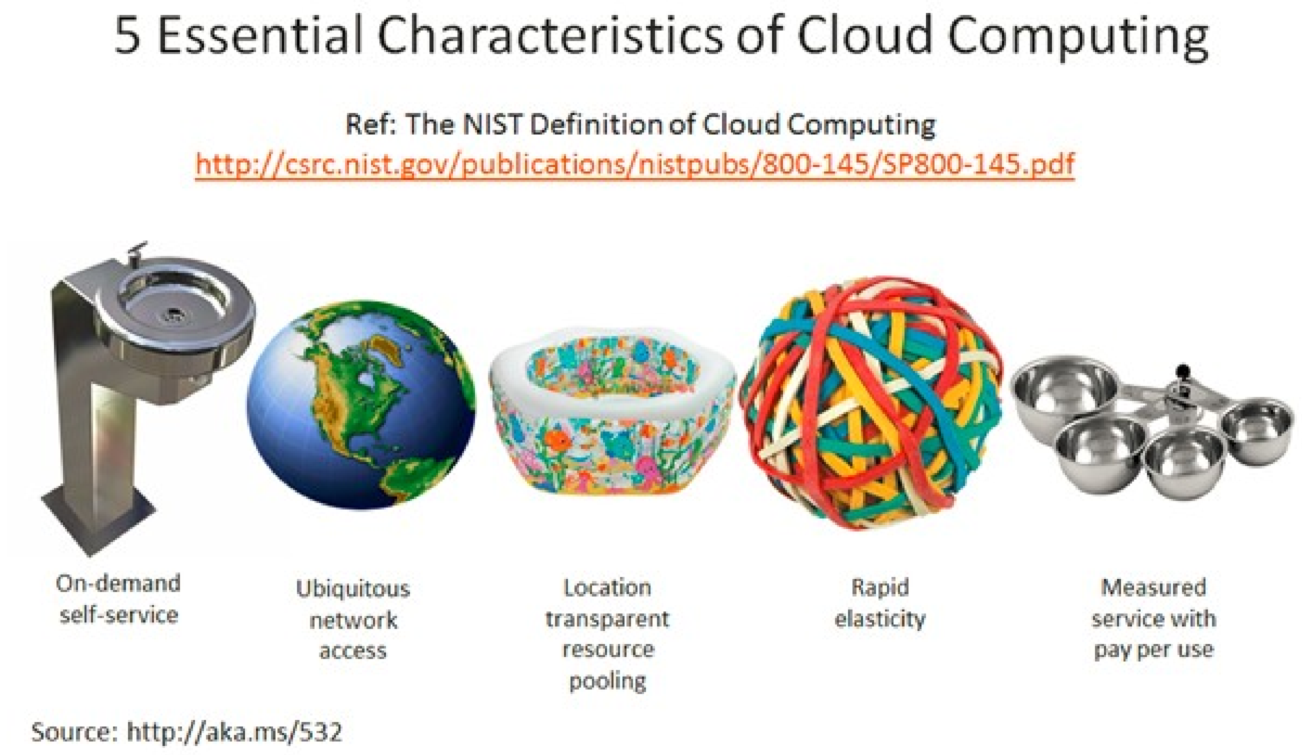
NIST specifies 3-4-5 rule of Cloud Computing

3 cloud service models or service types for any cloud platform

4 deployment models

5 essential characteristics of cloud computing infrastructure

Characteristics of Cloud Computing



5 Essential Characteristics of Cloud Computing

Ref: The NIST Definition of Cloud Computing
http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

On-demand self-service

Ubiquitous network access

Location transparent resource pooling

Rapid elasticity

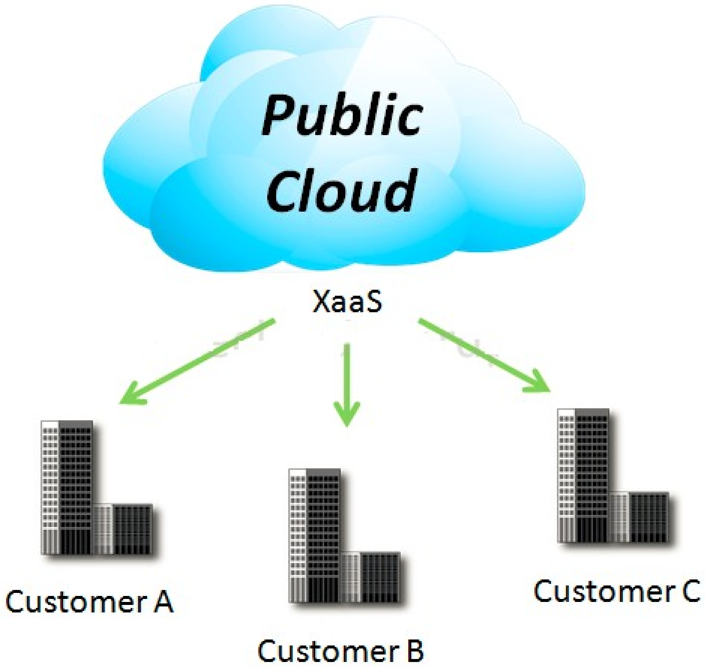
Measured service with pay per use

Source: http://aka.ms/532

- On demand self-service
- Broad network access
- Resource pooling
- Rapid elasticity
- Measured service

4 Deployment Models

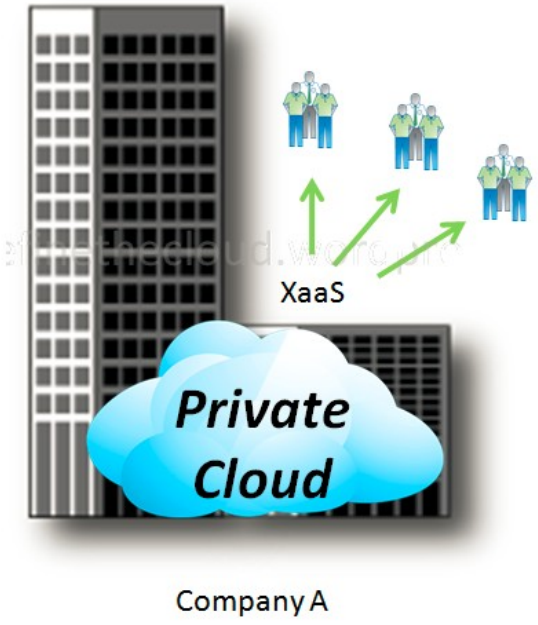
1. Public Cloud


Public Cloud
XaaS
Customer A
Customer B
Customer C

Mega-scale cloud infrastructure is made available to the general public or a large industry group and is owned by an organization selling cloud services.

4 Deployment Models

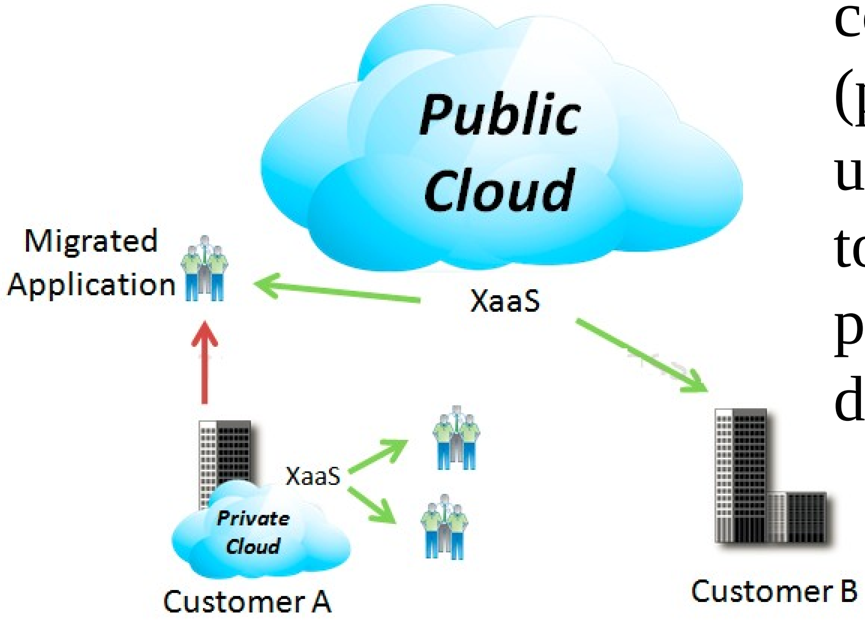
2. Private Cloud


XaaS
Private Cloud
Company A

The cloud infrastructure is operated solely for an organization. It may be managed by the organization or a third party and may exist on premise or off premise.

4 Deployment Models

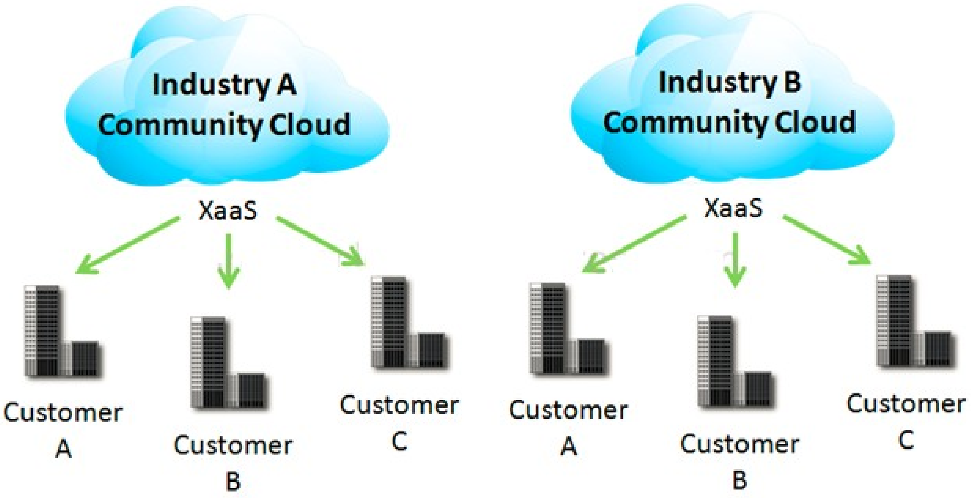
3. Hybrid Cloud


Public Cloud
Migrated Application
XaaS
XaaS
Private Cloud
Customer A
Customer B

The cloud infrastructure is a composition of two or more clouds (private or public) that remain unique entities but are bound together by standardized or proprietary technology that enables data and application portability

4 Deployment Models

4. Community Cloud


Industry A
Community Cloud
XaaS
Customer A
Customer B
Customer C
Industry B
Community Cloud
XaaS
Customer A
Customer B
Customer C

Community Clouds are when an ‘infrastructure is shared by several organizations and supports a specific community that has shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be managed by the organizations or a third party and may exist on premise or off premise’ according to NIST.

A community cloud is a cloud service shared between multiple organizations with a common tie/goal/objective.

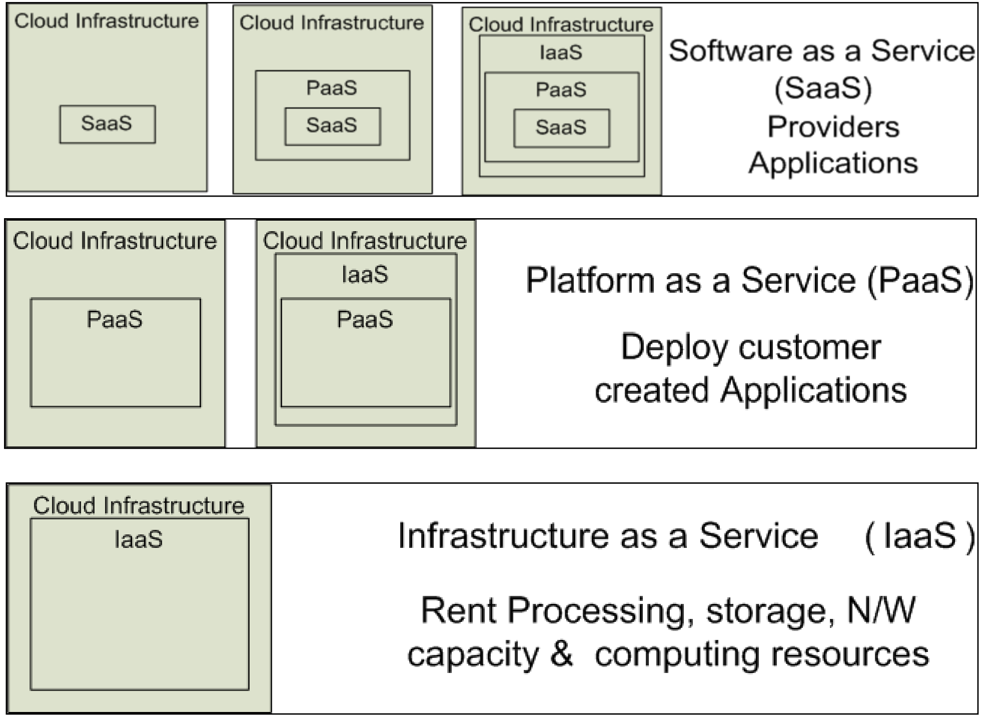
E.g. OpenCirrus

3 Cloud Service Models

Software as a Service

Platform as a Service

Infrastructure as a Service


Cloud Infrastructure
SaaS
Cloud Infrastructure
PaaS
SaaS
Cloud Infrastructure
IaaS
PaaS
SaaS
Cloud Infrastructure
PaaS
Cloud Infrastructure
IaaS
PaaS
Cloud Infrastructure
IaaS

SalesForce
CRM
LotusLive

Software as a Service (SaaS)
Providers Applications

Google App Engine

Windows Azure
The Future Made Familiar

Platform as a Service (PaaS)

Deploy customer created Applications

amazon web services™

rackspace® HOSTING

Infrastructure as a Service (IaaS)

Rent Processing, storage, N/W capacity & computing resources

Software as a Service (SaaS)

Software as a service features a complete application
offered as a service on demand.
A single instance of the software runs on the cloud and services multiple end users or client organizations.
E.g. salesforce.com , Google Apps

Platform as a Service

Platform as a service encapsulates a layer of software and provides it as a service that can be used to build higher-level services.

2 Perspectives for PaaS :-

1. Producer:- Someone producing PaaS might produce a platform by integrating an OS, middleware, application software, and even a development environment that is then provided to a customer as a service.

2. Consumer:-Someone using PaaS would see an encapsulated service that is presented to them through an API. The customer interacts with the platform through the API, and the platform does what is necessary to manage and scale itself to provide a given level of service.

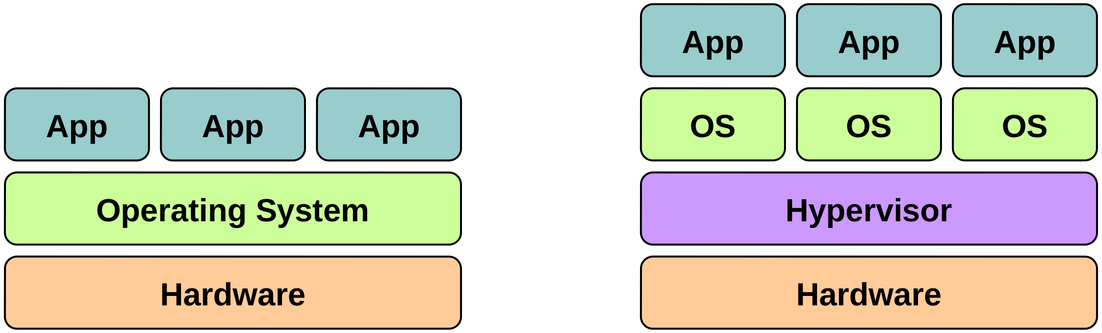
Virtual appliances can be classified as instances of PaaS.

Infrastructure as a Service

Infrastructure as a service delivers basic storage and computing capabilities as standardized services over the network.
Servers, storage systems, switches, routers , and other systems are pooled and made available to handle workloads that range from application components to high-performance computing applications.

Cloud Infrastructures

Key Technology is Virtualization


App App App
Operating System
Hardware

App App App
OS OS OS
Hypervisor
Hardware

Virtualization plays an important role as an enabling technology for datacentre implementation by abstracting compute, network, and storage service platforms from the underlying physical hardware

Cloud Providers Characteristics

- Provide on-demand provisioning of computational resources
- Use virtualization technologies to lease these resources
- Provide public and simple remote interfaces to manage those resources
- Use a pay-as-you-go cost model, typically charging by the hour
- Operate data centers large enough to provide a seemingly unlimited amount of resources to their clients

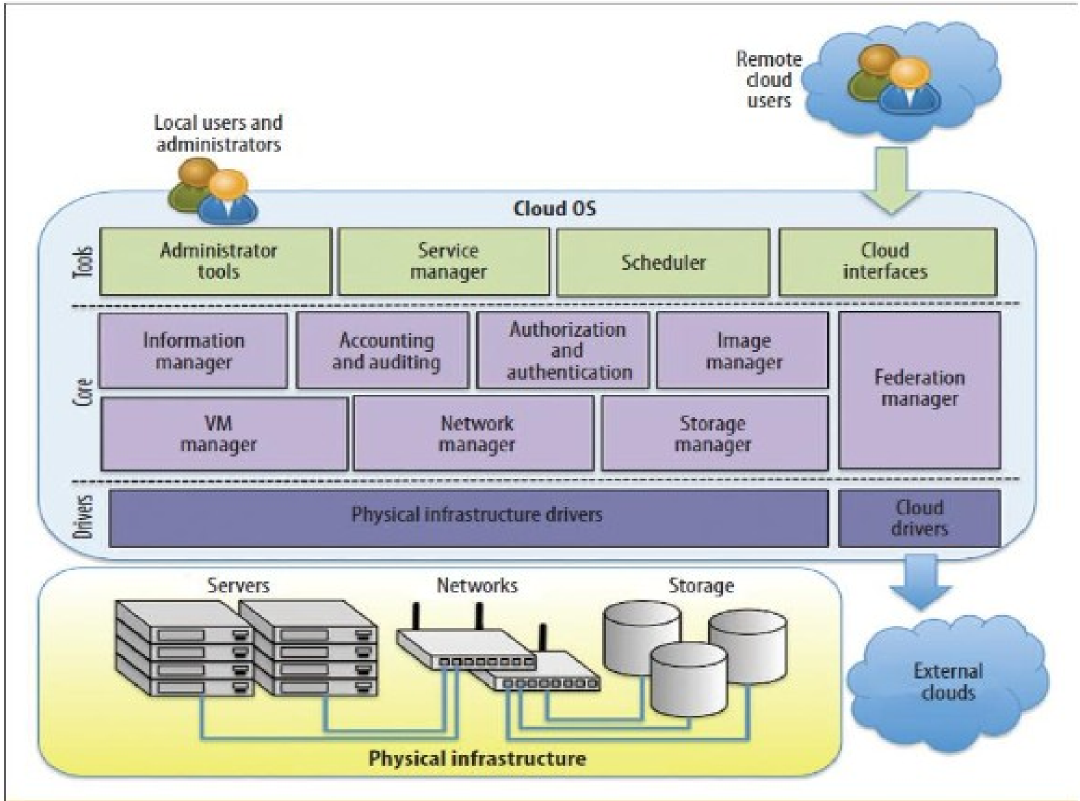
Management of Virtualized Resources

Distributed Management of Virtual Machines

Reservation-Based Provisioning of Virtualized Resources

Provisioning to Meet SLA Commitments

The Cloud OS


Remote cloud users
Local users and administrators
Cloud OS
Tools
Administrator tools
Service manager
Scheduler
Cloud interfaces
Core
Information manager
Accounting and auditing
Authorization and authentication
Image manager
Federation manager
VM manager
Network manager
Storage manager
Drivers
Physical infrastructure drivers
Cloud drivers
Servers
Networks
Storage
External clouds
Physical infrastructure

The cloud OS, the main component of an IaaS cloud architecture, is organized in three layers: drivers, core components, and high-level tools.

The cloud operating system is responsible for:
1. managing the physical and virtual infrastructure,
2. orchestrating and commanding service provisioning and deployment
3. providing federation capabilities for accessing and deploying virtual resources in remote cloud infrastructures

THANK YOU

BITS Pilani
Pilani Campus

BITS Pilani
Pilani Campus

DATABASE DESIGN

SESSION 1

innovate achieve lead

BITS Pilani
Pilani Campus

INTRODUCTION TO DATABASE SYSTEMS

Learning Objectives

- Basic terminologies in Database
- Advantages of data base system over conventional file system
- Architecture of data base system
- Three schema levels of data base system

Basic terminologies

Database and Database system

An organization contains different set of data.

The organization must have accurate and reliable data for effective decision making.

An organization can not run without data.

So the data in the organization should be maintained properly.

Database is used to maintain the data in proper way.

A data base is a collection of data and a database system is computerized record keeping system.

The data are stored in the secondary storage device (permanent Storage) in the form of files, tables, objects, etc.

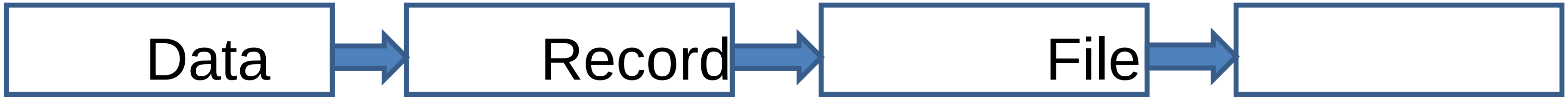
Basic terminologies

A **file** is a collection of one or more record(s) where a **record** is a collection of field values / attribute value.

A **field** is the lowest level of data item of a record or an entity.

The collection of inter-related files is called as the database.

The hierarchy is represented as follows:



Data Record File

Database

Basic terminologies

A single database may contain more than one heterogeneous files.

Example:

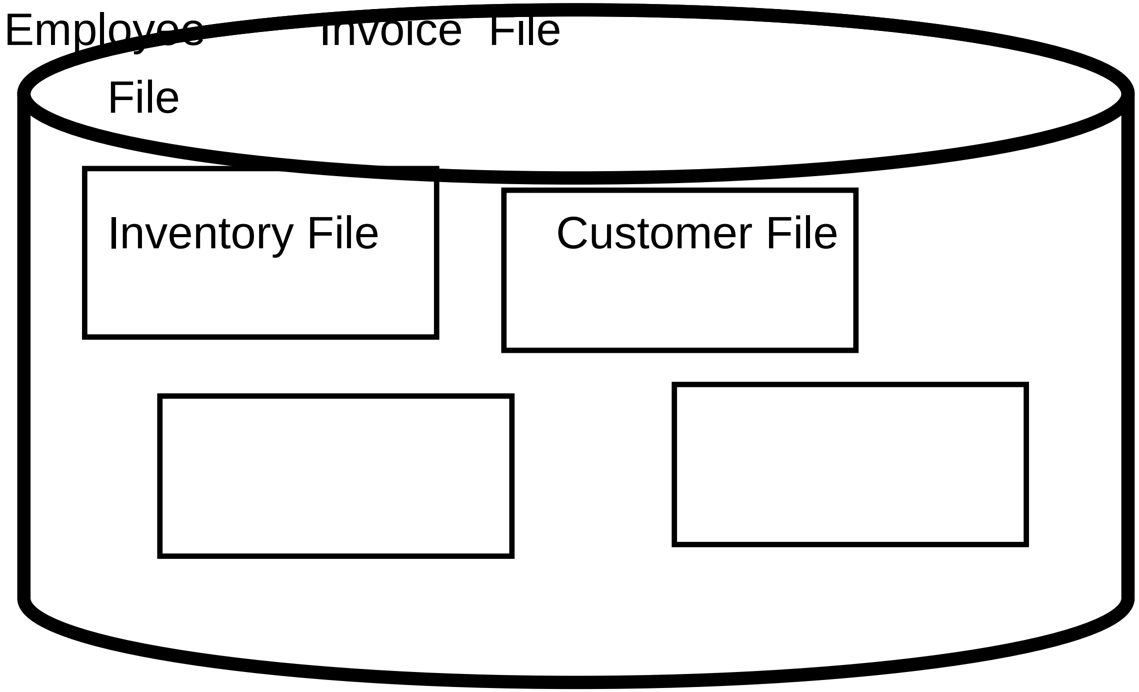
The database for a manufacturing company contains the files as

1. Employee file – It is a collection of records of all the employees working in the company
2. Customer file - It is a collection of records of all the customers of the company
3. Inventory file – It is a collection of records of maintaining the stock

Basic terminologies

4. Invoice file – It is a collection of records of bills of the customers

The database of the company can be represented as follows



Employee File

Invoice File

Inventory File

Customer File

Basic terminologies

Database management System – The software which is used to manage the data in the database is called as the database management system.

The management represents the following

1. (i) Creating a new database for collecting the data
2. (ii) Adding new data into the existing database
3. (iii) Deleting existing data in the database
4. (iv) Modifying an existing data in the database
5. (v) Listing data of the database with or without using a condition.

Advantages of Database System

1. **Redundancy can be reduced**

When data are redundant, we face two problems.

They are

  i. Wastage of spaces in the disk

  ii. Inconsistency problem

In data base system, redundancy can be avoided with a mechanism called as Normalization.

Normalization is the procedure which removes the unwanted duplicate data by splitting the table into two or more tables. For Eg.

Assume that a table consists of the data of the students who are doing a particular course.

Advantages of Database System

Student table

Stu#	StuName	dept#	Coun#	CounName
S101	Ram	D1	ST002	Dr. Jackson
S102	Bala	D2	ST002	Dr.Jackson
S111	Madhu	D1	ST001	Dr. Venkat
S122	Divya	D1	ST001	Dr. Venkat
S115	Hema	D2	ST001	Dr. Venkat
S103	Sachin	D1	ST001	Dr. Venkat

Coun# - Counselor id   CounName – Counselor name

Advantages of Database System

The above table is split into two tables (Student and Staff) as follows:

Student

Stu#	StuName	dept#	Coun#
S101	Ram	D1	ST002
S102	Bala	D2	ST002
S111	Madhu	D1	ST001
S122	Divya	D1	ST001
S115	Hema	D2	ST001
S103	Sachin	D1	ST001

Staff

Coun#	CounName
ST001	Dr. Venkat
ST002	Dr. Jackson

Duplicate data related to staff is reduced

Advantages of Database System

2.Inconsistency is avoided

If the value of the attribute shows similar value when it is accessed by two different applications, then data is said to be Consistent.

Inconsistency may arise if they are redundant.

When duplication is avoided, there is no possibility of Inconsistency (Single copy has a single value only).

Though multiple copies are available and when a transaction accesses any one of the copy, a procedure in data base system called as “Propagating updates“ updates the latest value in all the copies.

It avoids inconsistency.

Advantages of Database System

3.Data can be shared

Concurrently, data can be accessed by more than one application.

Database which share data to multiple applications is called as the distributed data base.

Distributed data base creates multiple mirror image copies of data and are shared with different applications that are executed in different locations.

When any modification is done on the mirror image copies, they automatically updated in the original copy of the data.

Example: Railway reservation system shares the data to different reservation counters.

Advantages of Database System

4.Data can be integrated

Data that exist in different files can be integrated (Combined) to answer the query of an application.

Example:

Assume that tables Student and Staff consist of the data about the students and staff who handle various courses.

To answer a query , “List the students who have scored more than 90% of marks in various subjects and the information of the staff who handled the subject”, we need the data from both the tables (Student and Staff)

Advantages of Database System

5. Security mechanism can be applied

Confidential data should be secured from unauthorized users.

To protect them from unauthenticated access, data base systems support a security mechanism known as Control access mechanism.

The control mechanism is divided into two types such as Software Controlling mechanism and hardware control mechanism.

Software Control mechanism – Data are protected with the help of a software.

Advantages of Database System

Examples:

a. **Authentication** procedure which asks login name and password to distinguish authorized and others.

b. **Encryption** procedure which encrypts the data into unknown format known as Cipher text so that it can be understood by the intruders

Hardware Control mechanism – Data are permitted to be accessed if the user should use physical component.

Example:

Usage of Access Entry card to enter into the prohibited area.

Advantages of Database System

6.Integrity can be maintained

The problem of integrity is the problem of ensuring that data in the database is correct.

When the input values (data) are given to be added into the database, there is a possibility of giving invalid values.

Invalid values create a lot of problem in the data base management.

Some conditions called as integrity constraints in Database system can be enforced at the time of creating the schema of the table.

These constraints check whether the inputs given by the user are valid or not.

Advantages of Database System

If they are invalid, the constraints produce appropriate error messages without storing the invalid values in to the table.

So there is no possibility of keeping Invalid values in the table.

Example: Assume that a table is created for keeping the information about employees working in hourly basis and one of the field is HOURS_WORKED which is used to calculate the wage.

We know that the field may have the value from 0 to 24.

If invalid value is given for the field constructing the record, database system displays the error message if integrity constraints are set.

Advantages of Database System

7.Conflicting requirements can be balanced

We know that data base can be accessed by more than one users and applications.

They may require different storage structures and access methods.

DBA chooses the best storage structure and access method so that performance of the system is improved.

8.Data Independence

In the conventional file system, all the database are dependent data.

Advantages of Database System

When the modification is to be performed on the storage structure, or if we need to make changes in the application of the data base also, the data is said to be dependent data.

If the application does not require any updating, the data is said to be **independent data**. Database system supports data dependence.

There are two types of data independence. They are

1. Logical data independence
2. Physical data independence

Advantages of Database System

Logical data independence - When the modification is performed on the application of the data base, If we do not need any change in the storage structure, then the data dependence is called as Logical data independence

Physical data independence - When the modification is performed on the storage structure, If we do not need any change in the application of the data base, data dependence is called as Physical data independence

Architecture of Database system

Components of a database system

1. **Data** – Data which can be shared between multiple applications and integrated for a single application.

Example for Sharing, the Faculty file can be accessed by the HR department application related to the promotion or for doing some other HR related operations.

Same data can be accessed by the application of the Finance department to prepare payroll.

Example for integrating, Faculty and student files are to be accessed for preparing the list which requires data of both students and Faculty member.

Architecture of Database system

2. Hardware – The hardware components of the system that consists of

a. SSD (Secondary Storage Device) such as Magnetic tape, Disk to hold data permanently

b. Processors for executing data(ALU and Control Unit)

c. Main memory for executing and storing data temporarily

d. Network – the mechanism to connect more than one Computers

3. Software – It is a software which is used for performing the operation on the database.

It is called as the database management system (DBMS).

Architecture of Database system

The requests from the users to access the data in the database like adding / removing data, retrieving data and updating data are handled by the DBMS.

4. Users – The users in the database system are classified into four types as Application Programmers, Naïve users, Sophisticated Users and database administrator.

Application Programmers – They are the computer professionals who write the application program in a database language or a programming language to access the data in the database.

Architecture of Database system

Naïve Users are unsophisticated users who interact with the system by invoking one of the application programs that have been written by the application programmer.

Sophisticated Users are the users who interact without writing programs.

Instead, they form their requests in a query language.

They submit the query to the tool called as query processor.

The query processor breaks the query into multiple instructions that the storage manager can understand.

Architecture of Database system

Database Administrator (DBA) is the person who has a central control over the data and the programs that can access the data.

The functions of the DBA include

- **Schema definition** – Schema represents the overall structure of the database.

DBA is responsible for deciding what fields are to be used, their data types and their length.

- **Storage structure and access method** – DBA is responsible for deciding how the data are to be stored in the database (disk) and the way in which the data can be accessed (Sequential and random)

Architecture of Database system

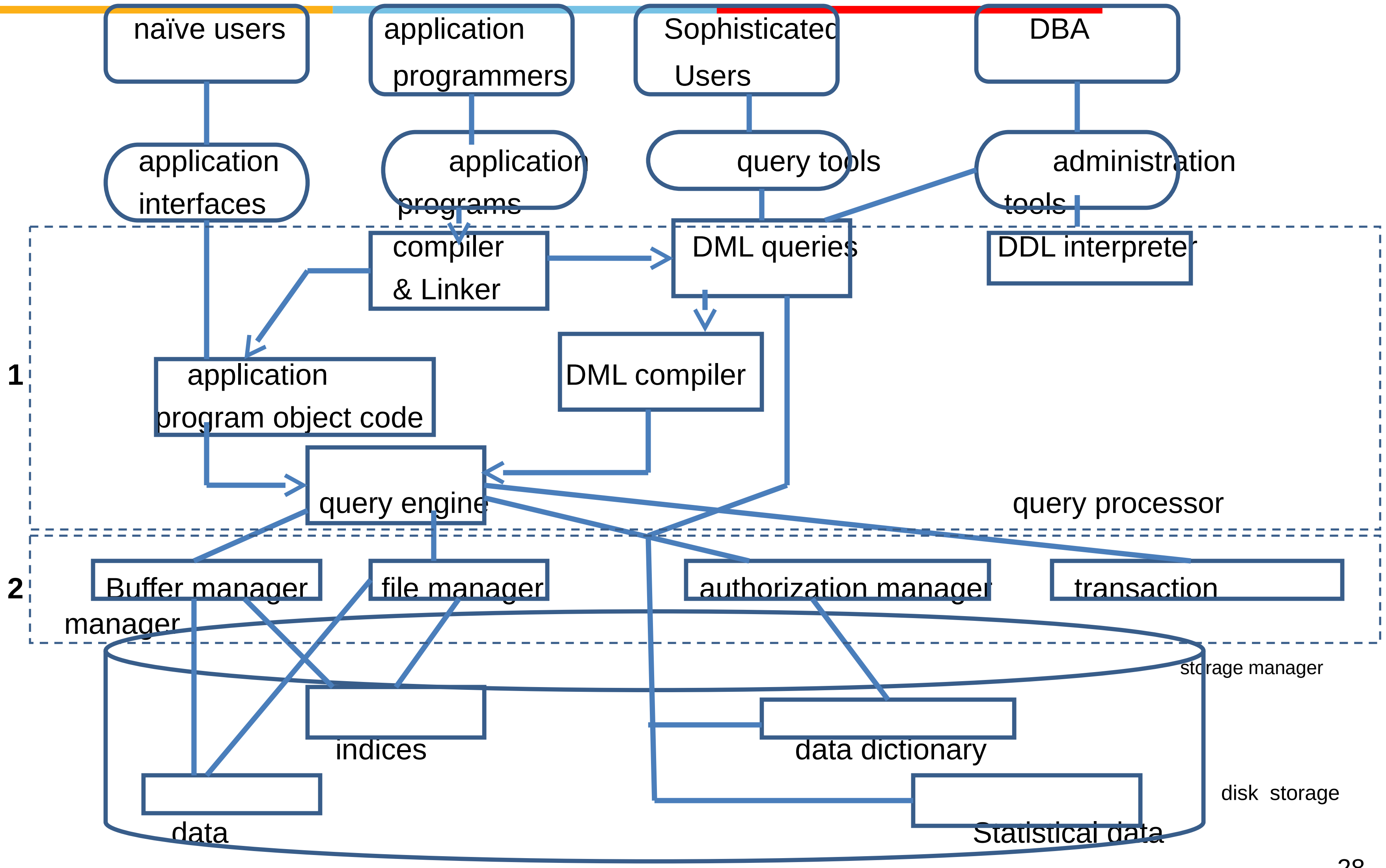
- **Modification in Schema definition and storage structure –**

DBA is responsible for making any changes in the data structure of the database and storage structure

- **Granting authorization for data access** – DBA has the rights to grant the permission to the users who are authorized to access the data
- **Performing periodical operations** – DBA is responsible for performing routine maintenance activities such as

- Doing back up periodically for preventing loss of data
- Monitoring the performance
- Ensuring that enough free disk space is available

Architecture of Database system


naïve users
application programmers
Sophisticated Users
DBA
application interfaces
application programs
query tools
administration tools
compiler & Linker
DML queries
DDL interpreter
1
application program object code
DML compiler
query engine
query processor
2
Buffer manager
file manager
authorization manager
transaction manager
storage manager
indices
data dictionary
disk storage
data
Statistical data

Architecture of Database system

5. Other components

The systems has some more components such as indices, data dictionary and Statistical data.

- *Indices* are the tables which are used for accessing the records randomly
- *Data dictionary* is also table which is the data about data.

It contains information such as type, length, etc., about the data stored in the database

- *Statistical data* – These are the data which contain statistical information of the data (How often they are accessed , who had accessed frequently, etc.,)

DBMS Schemas: Internal, Conceptual, External

Database systems comprise of complex data structures.

Thus, to make the system efficient for retrieval of data and reduce the complexity of the users, developers use the method of Data Abstraction.

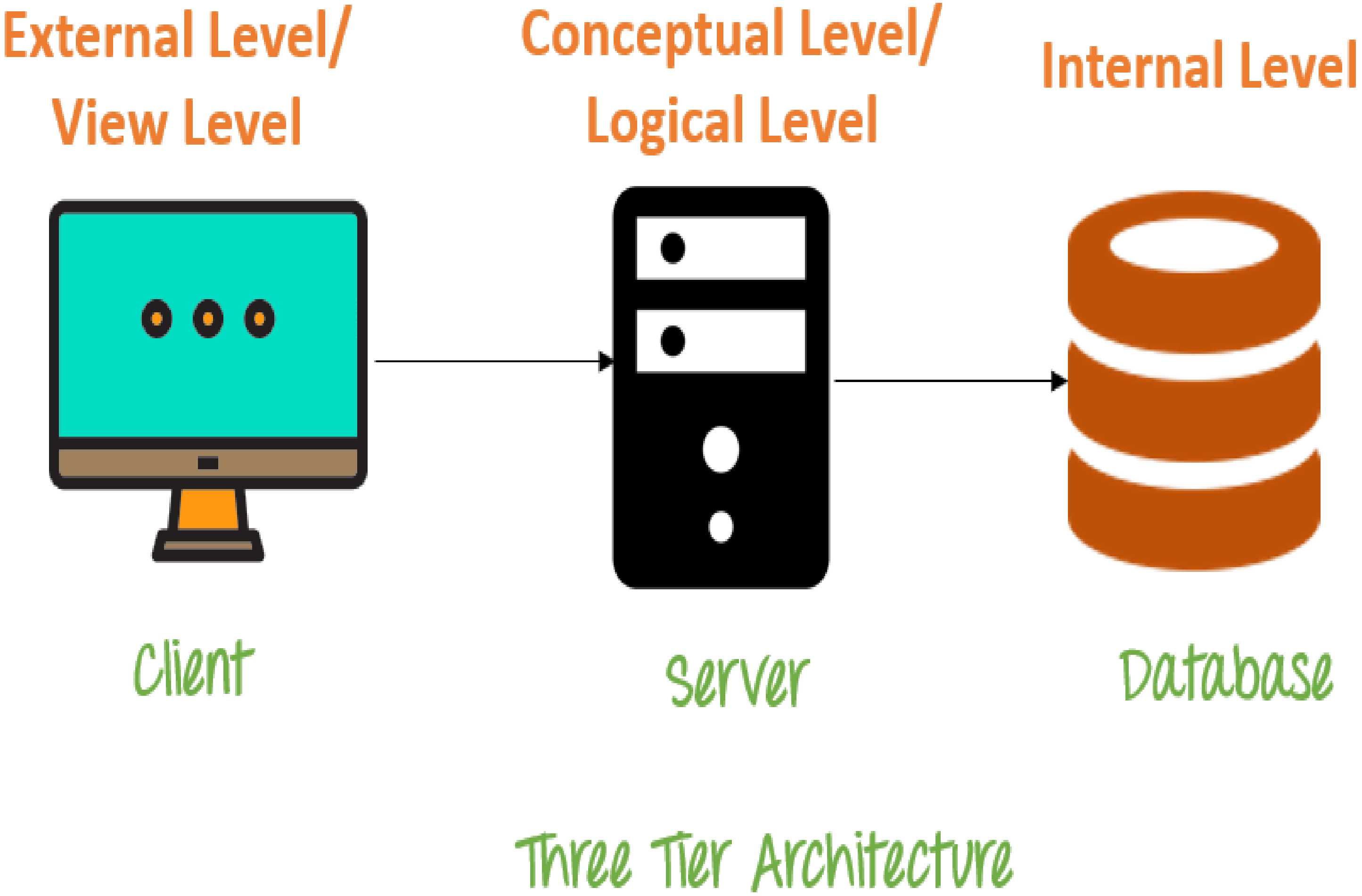
There are mainly three levels of data abstraction:

Internal Level: Actual PHYSICAL storage structure and access paths.

Conceptual or Logical Level: Structure and constraints for the entire database

External or View level: Describes various user views

DBMS Schemas: Internal, Conceptual, External



External Level/ View Level

Conceptual Level/ Logical Level

Internal Level

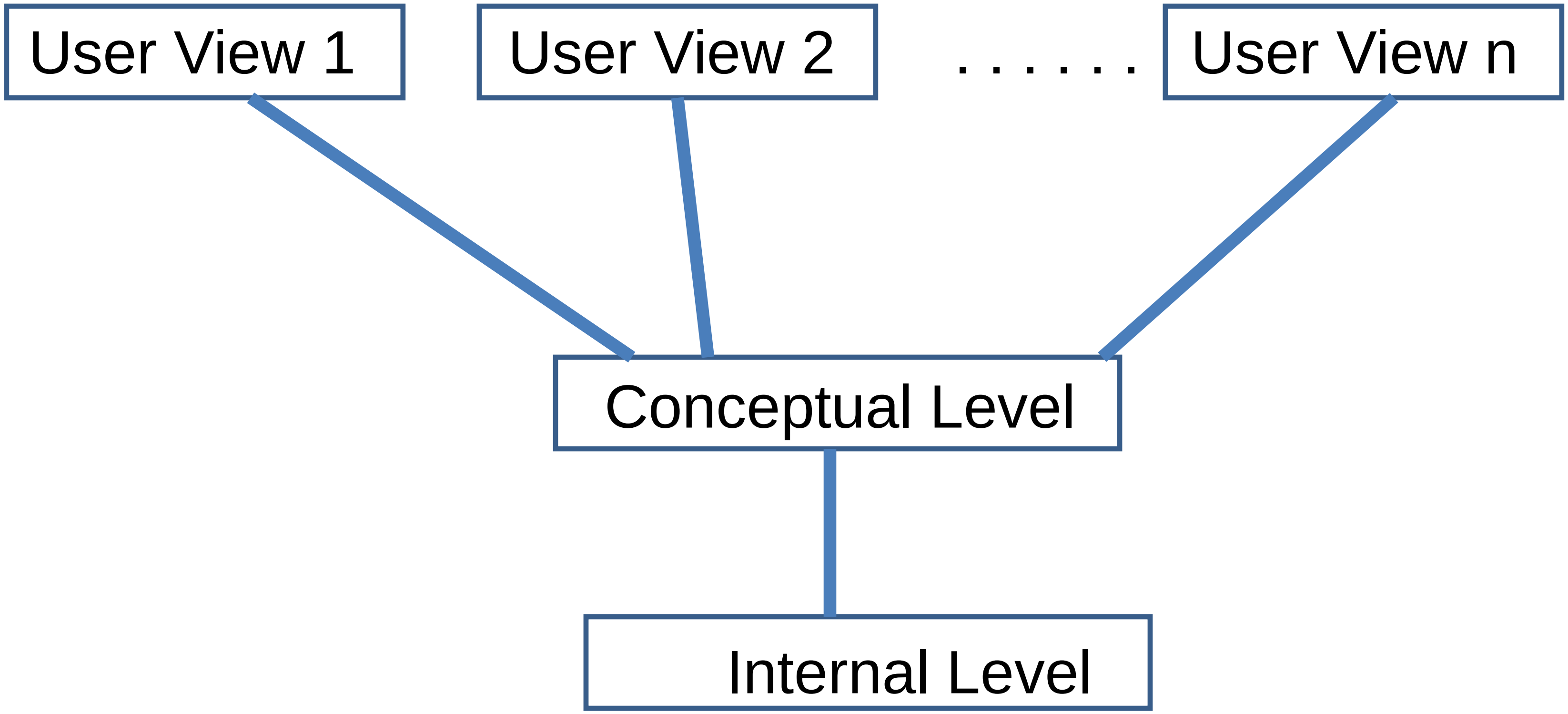
Client

Server

Database

Three Tier Architecture

DBMS Schemas: Internal, Conceptual, External



User View 1

User View 2

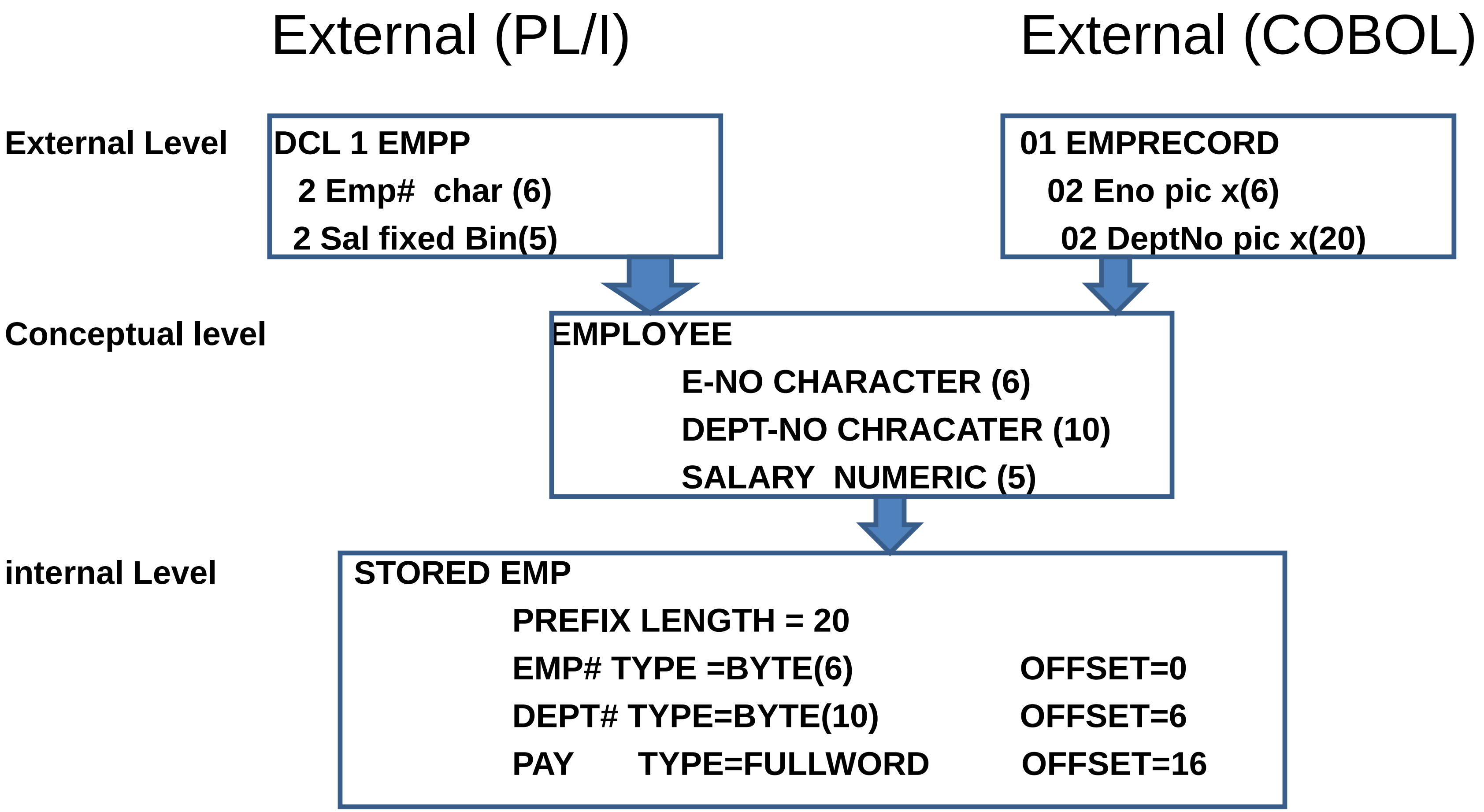
. . . .

User View n

Conceptual Level

Internal Level

DBMS Schemas: Internal, Conceptual, External



External (PL/I)

External (COBOL)

External Level

```
DCL 1 EMPP
  2 Emp#  char (6)
 2 Sal fixed Bin(5)
```

```
01 EMPRECORD
   02 Eno pic x(6)
     02 DeptNo pic x(20)
```

Conceptual level

```
EMPLOYEE
              E-NO CHARACTER (6)
              DEPT-NO CHRACATER (10)
              SALARY  NUMERIC (5)
```

internal Level

```
STORED EMP
                PREFIX LENGTH = 20
                EMP# TYPE =BYTE(6)              OFFSET=0
                DEPT# TYPE=BYTE(10)             OFFSET=6
                PAY      TYPE=FULLWORD          OFFSET=16
```

DBMS Schemas: Internal, Conceptual, External

The purpose of each level is given below

Internal Level/Schema

The internal schema specifies the physical storage structure of the database.

The internal schema is a very low-level representation of the entire database.

It contains multiple occurrences of multiple types of internal record.

These records are also called "stored records”.

Facts about Internal schema:

- The internal schema is the lowest level of data abstraction

DBMS Schemas: Internal, Conceptual, External

- It helps us to keep information about the actual representation of the entire database.
- It is like the actual storage of the data on the disk in the form of records
- The internal view tells us what data is stored in the database and how they are stored
- It never deals with the physical devices.
- Instead, internal schema views a physical device as a collection of physical pages

Conceptual Schema/Level

The conceptual schema mentions the Database structure of the whole database for the community of users.

DBMS Schemas: Internal, Conceptual, External

This schema hides information about the physical storage structures and focuses on describing data types, entities, relationships, etc.

This logical level lies between the user level and physical storage view.

However, there is only single conceptual view of a single database.

Facts about Conceptual schema:

- Defines all database entities, their attributes, and their relationships
- Security and integrity information

DBMS Schemas: Internal, Conceptual, External

- In the conceptual level, the data available to a user must be contained in or derivable from the physical level

External Schema/Level

An external schema specifies the part of the database which specific user is interested in.

It hides the unrelated details of the database from the user.

There may be "n" number of external views for each database.

Each external view is defined using an external schema, which consists of definitions of various types of external record of that specific view.

DBMS Schemas: Internal, Conceptual, External

An external view is just the content of the database as it is seen by some specific particular user.

For example, a user from the sales department will see only sales related data.

Facts about external schema:

- An external level is only related to the data which is viewed by specific end users.
- This level includes some external schemas.
- External schema level is nearest to the user
- The external schema describes the segment of the database which is needed for a certain user group and hides the remaining details from the database from the specific user group.

DBMS Schemas: Internal, Conceptual, External

Goal of three level schema of Database

Here, are some Objectives of using Three schema Architecture:

- Every user should be able to access the same data but able to see a customized view of the data.
- The user need not to deal directly with physical database storage detail.
- The DBA should be able to change the database storage structure without disturbing the user's views.
- The internal structure of the database should remain unaffected when changes made to the physical aspects of storage.

DBMS Schemas: Internal, Conceptual, External

Advantages of Database Schema

- We can manage data independent of the physical storage
- Faster Migration to new graphical environments
- DBMS Architecture allows us to make changes on the presentation level without affecting the other two layers
- As each tier is separate, it is possible to use different sets of developers.
- It is more secure as the client doesn't have direct access to the database business logic
- In case of failure of one-tier, no data loss is acquired, as we are always secure by accessing the other tier.

DBMS Schemas: Internal, Conceptual, External

Disadvantages Database Schema

- Complete DB Schema is a complex structure which is difficult to understand
- Difficult to set up and maintain
- The physical separation of the tiers can affect the performance of the Database

Thanks

Database Design

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Contact Session 2: Data Modelling

What are we dealing today?

Database Design and ER Modelling

1. Steps in database Design Process
2. Concepts and notations
3. Relationships and constraints
4. Examples

Entity Relationship Model

ERD

- It describes the data involved in a real world enterprise in terms of objects and their relationship
- It illustrates the logical structure of databases .
- It is used in a phase called conceptual database design
- ER model is used mainly as a design tool and documentation for the system
- Peter Chen developed ERDs in 1976.

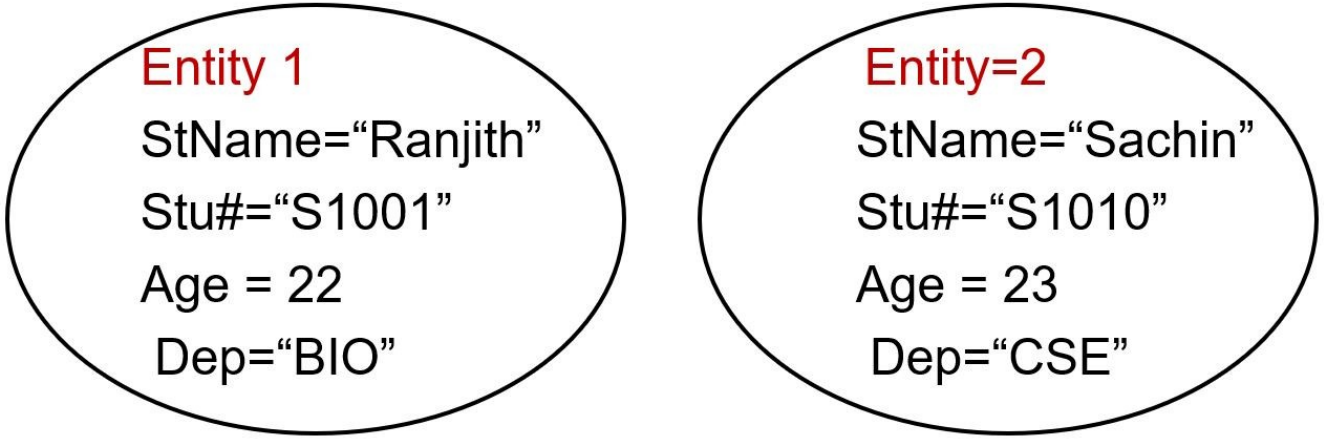
Entity, Entity Set Attributes

- An entity is an object or concept about which you want to store information.
- An entity may be an object with a physical existence (person, car) or with a conceptual existence (company, university).
- An entity has properties which are known as attributes of the entity.
- Attributes can be simple or composite.

Examples:

An entity STUDENT has the following attributes

StName (Student Name), Stu# (Student Number) , age and dep(Department Name)

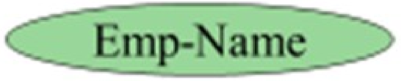


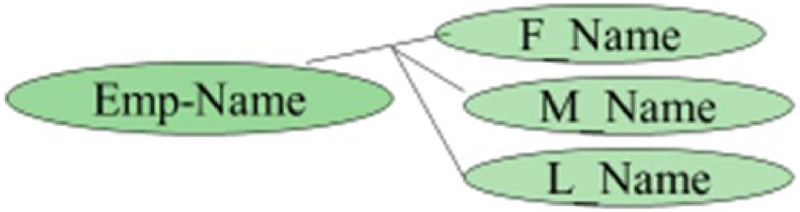
Entity 1
StName="Ranjith"
Stu#="S1001"
Age = 22
Dep="BIO"

Entity=2
StName="Sachin"
Stu#="S1010"
Age = 23
Dep="CSE"

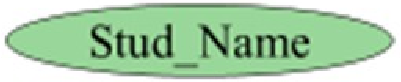
Attribute types

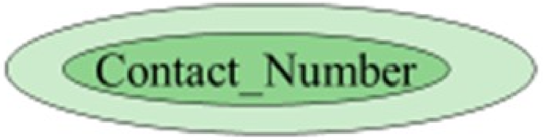
- Simple & Composite



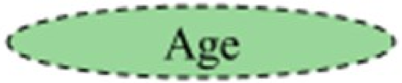

Emp-Name
F_Name
M_Name
L_Name

- Single-valued & Multi-valued



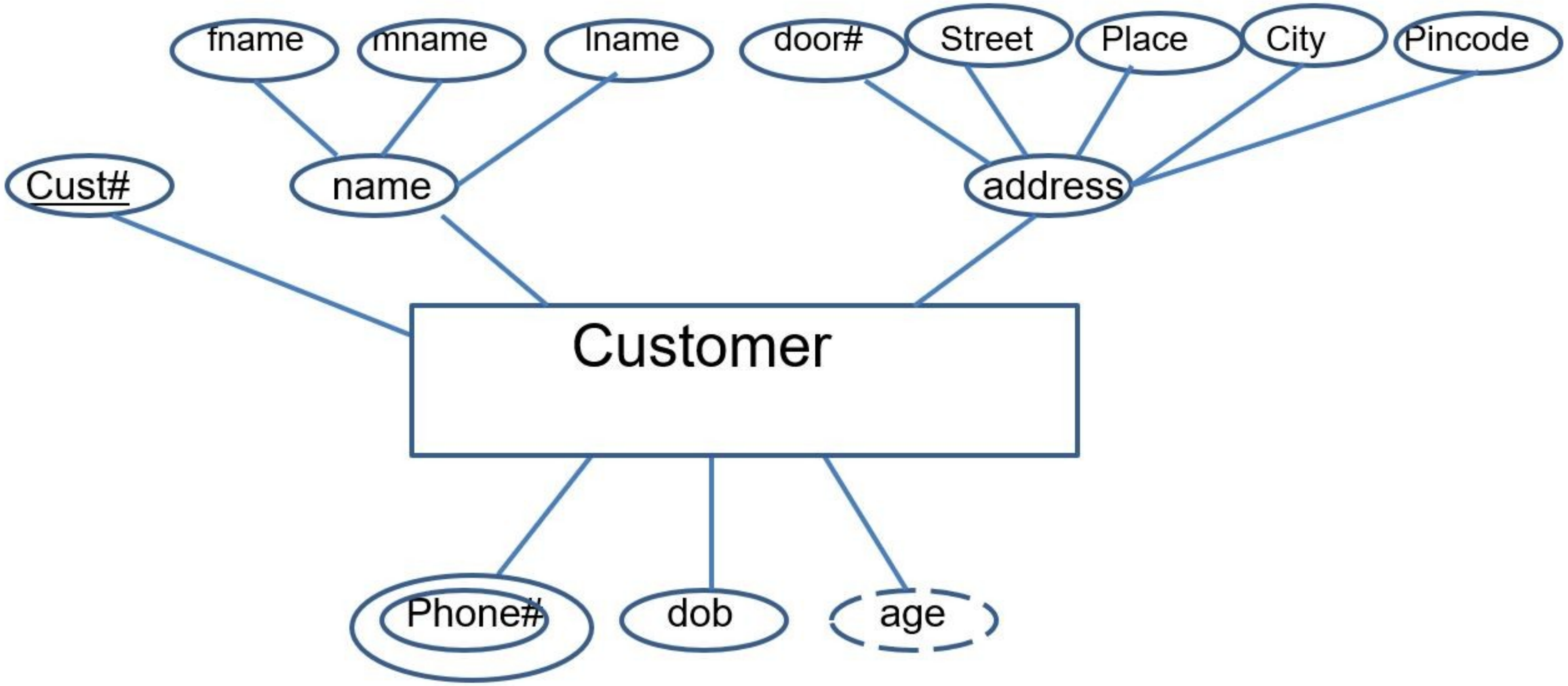

Contact_Number

- Derived
  - E.g Age: Age could be derived from date of Birth



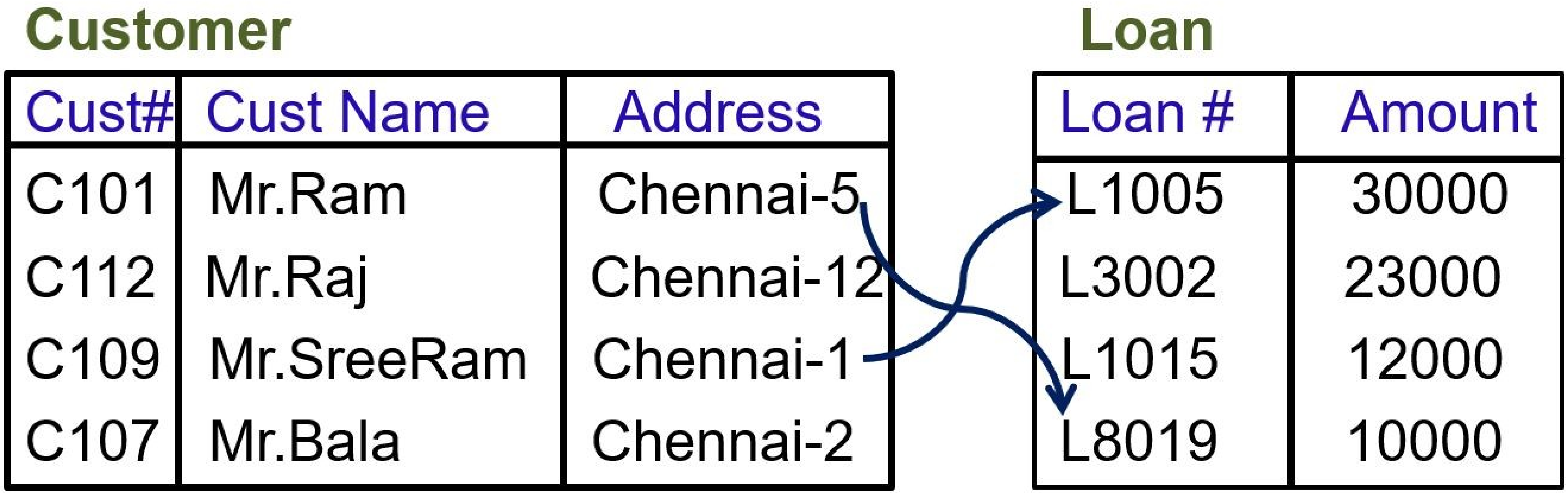
Entity Relationship Model

The following is the ER diagram that consists of different types of attributes


fname
mname
lname
door#
Street
Place
City
Pincode
Cust#
name
address
Customer
Phone#
dob
age

Entity Relationship Model

- **3. Relationship**
- A relationship represents the association between two or more entities.
- Example: Assume that there are two entity sets: Customer and Loan as



Customer

Cust#	Cust Name	Address
C101	Mr.Ram	Chennai-5
C112	Mr.Raj	Chennai-12
C109	Mr.SreeRam	Chennai-1
C107	Mr.Bala	Chennai-2

Loan

Loan #	Amount
L1005	30000
L3002	23000
L1015	12000
L8019	10000

Entity Relationship Model

In the above diagram , there are two relationships.

First relationship is defined that Customer Mr. Ram has taken a loan of Rs.10000 and its number is L8019.

The second relationship shows that Mr. SreeRam has received the loan amount of Rs.30000 and its loan number is L1005.

Entity set

It is a set of entities of the same type that share same properties or attributes.

Example: A set of persons who are customers at a particular bank can be defined as the entity set **Customer** as in the following diagram

Entity Relationship Model

Customer

Cust#	Cust Name	Address	Balance(Rs)
C101	Mr.Ram	Chennai-5	28235
C112	Mr.Raj	Chennai-12	3140
C109	Mr.SreeRam	Chennai-1	112
C107	Mr.Bala	Chennai-2	12345

Entity Relationship Model

Symbols used for drawing ER diagrams and their usages are :

Rectangles - To represent entity sets

Ellipses - To represent attributes

Diamonds - To represent the relationship set

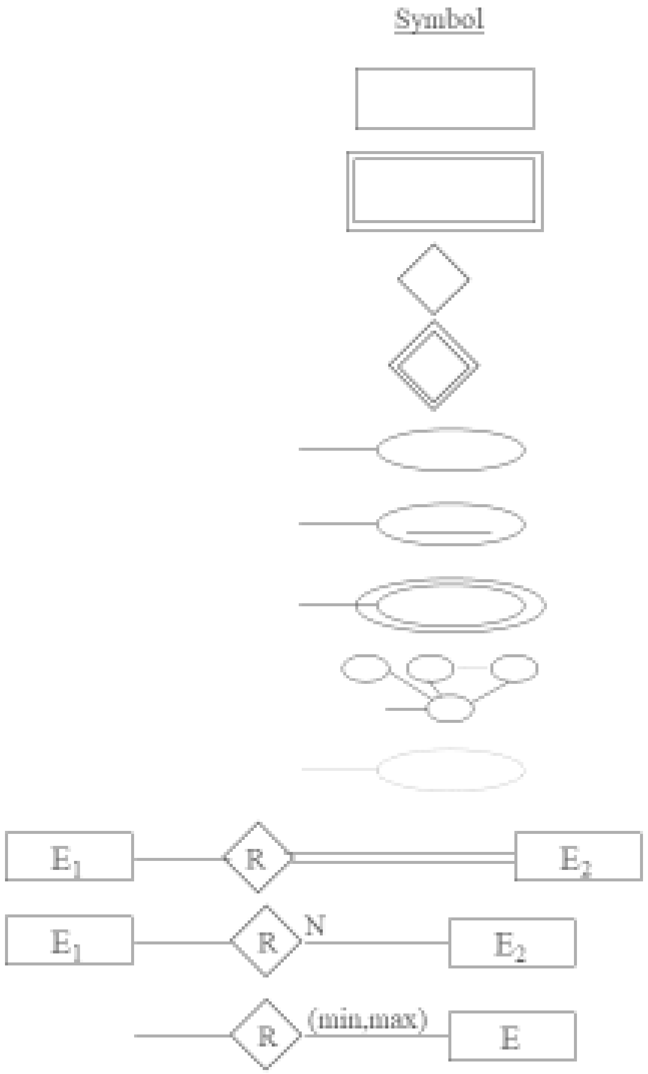
Line - To link attributes with the entity and entity to another entity

Double Ellipses – To represent multivalued attributes

Dotted Ellipses – To represent Derived attribute

Double rectangles – Representing weak entity sets

NOTATION FOR ER SCHEMAS



Meaning

ENTITY TYPE

WEAK ENTITY TYPE

RELATIONSHIP TYPE

IDENTIFYING RELATIONSHIP TYPE

ATTRIBUTE

KEY ATTRIBUTE

MULTIVALUED ATTRIBUTE

COMPOSITE ATTRIBUTE

DERIVED ATTRIBUTE

TOTAL PARTICIPATION OF E_2 IN R

CARDINALITY RATIO 1:N FOR E_1:E_2 IN R

STRUCTURAL CONSTRAINT (min, max) ON PARTICIPATION OF E IN R

ERD

- Identifying Entities

Given a system description you can identify entities by:

- Select all nouns
- Eliminate one’s not necessary
- Filter those for which system needs to store record.
- Identify the attributes that describes the entity

Entity

Entities are classified as:

- DOMINANT ENTITIES: An entity that does not depend on another entity for its existence is a dominant or strong entity
- WEAK ENTITIES: An entity that depends on some other entity for its existence

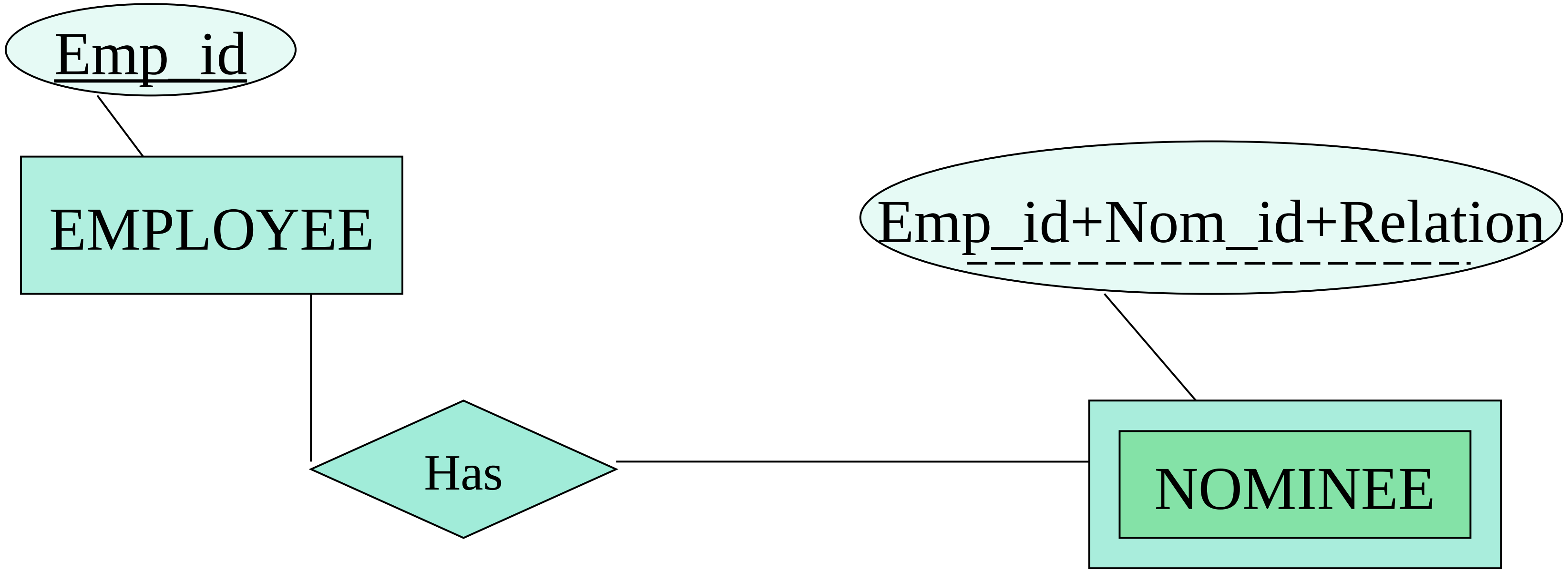
Existence Dependencies

- If the existence of entity x depends on the existence of entity y, x is said to be existence dependent on y.

- Entity y is said to be a Dominant entity and x is said to be a subordinate entity

Weak Entity

- A weak entity does not have a primary key of its own
- You can use a combination of the primary key of the dominant entity & an attribute of the weak entity as the primary key of the weak entity.
- Thus a discriminator of Weak Entity + Prime attribute of the its Dominant Entity.


Emp_id
EMPLOYEE
Emp_id+Nom_id+Relation
Has
NOMINEE

Relationships

- Entities are linked to each other through relationships.

Hence a relationship is an association between 2 entities.

- Eg:

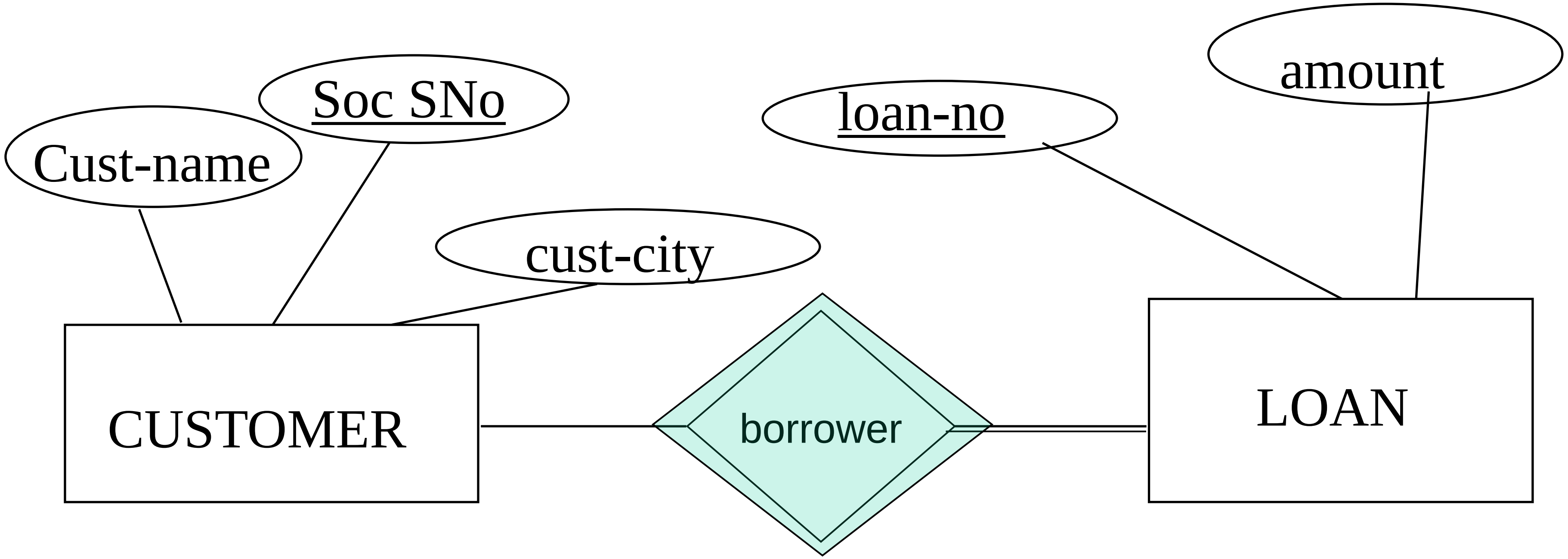
Entity	Relationship	Entity
SUPPLIER	supplies	ITEM

Participation of an entity set in a relationship

- Total: The participation of an entity set E in a relationship set R is said to be total if every entity in E participates in at least one relationship R

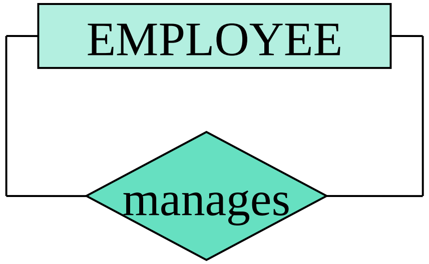
- Partial: Part of the set of entities participate in some relationship

Total & Partial Participation

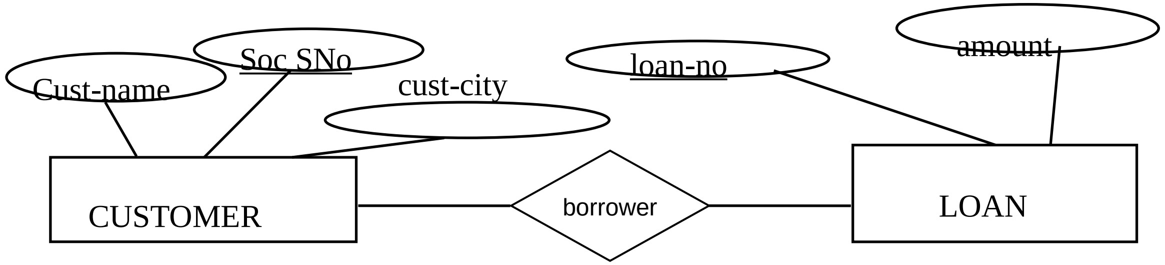

Soc SNo
Cust-name
cust-city
loan-no
amount
CUSTOMER
borrower
LOAN

Degree of Relationship

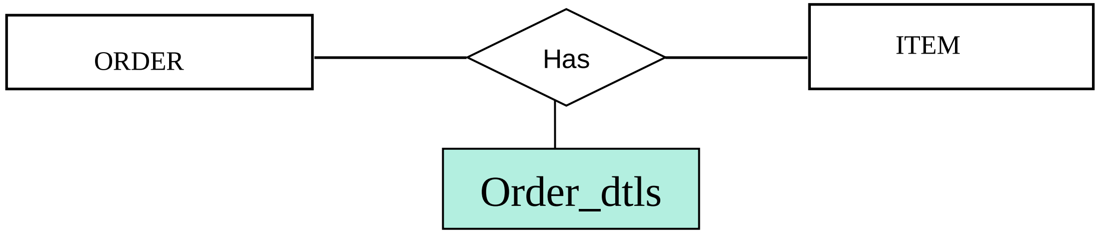
Unary : Degree 1


EMPLOYEE
manages

Binary : Degree 2


Cust-name
Soc SNo
cust-city
loan-no
amount
CUSTOMER
borrower
LOAN

Ternary : Degree 3

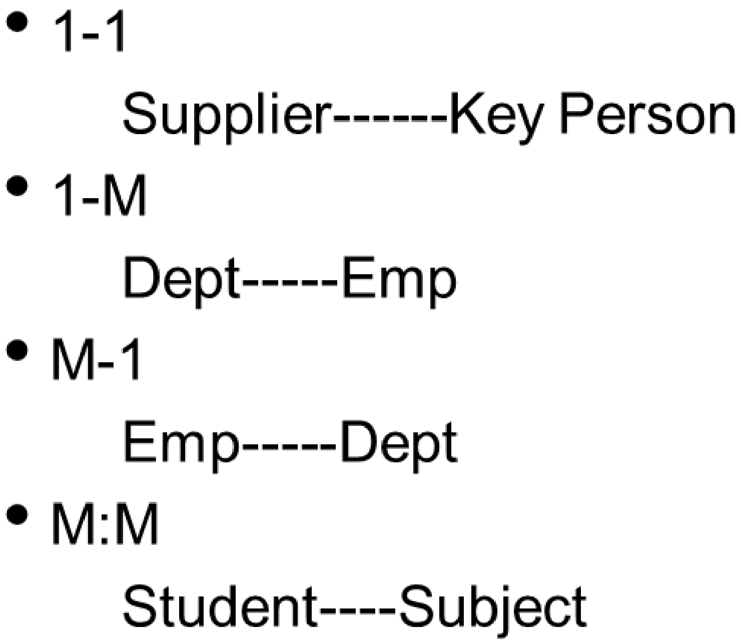

ORDER
Has
ITEM
Order_dtls

N-ary : Degree n

Further Reading: https://www.gatevidyalay.com/tag/total-and-partial-participation-in-dbms/

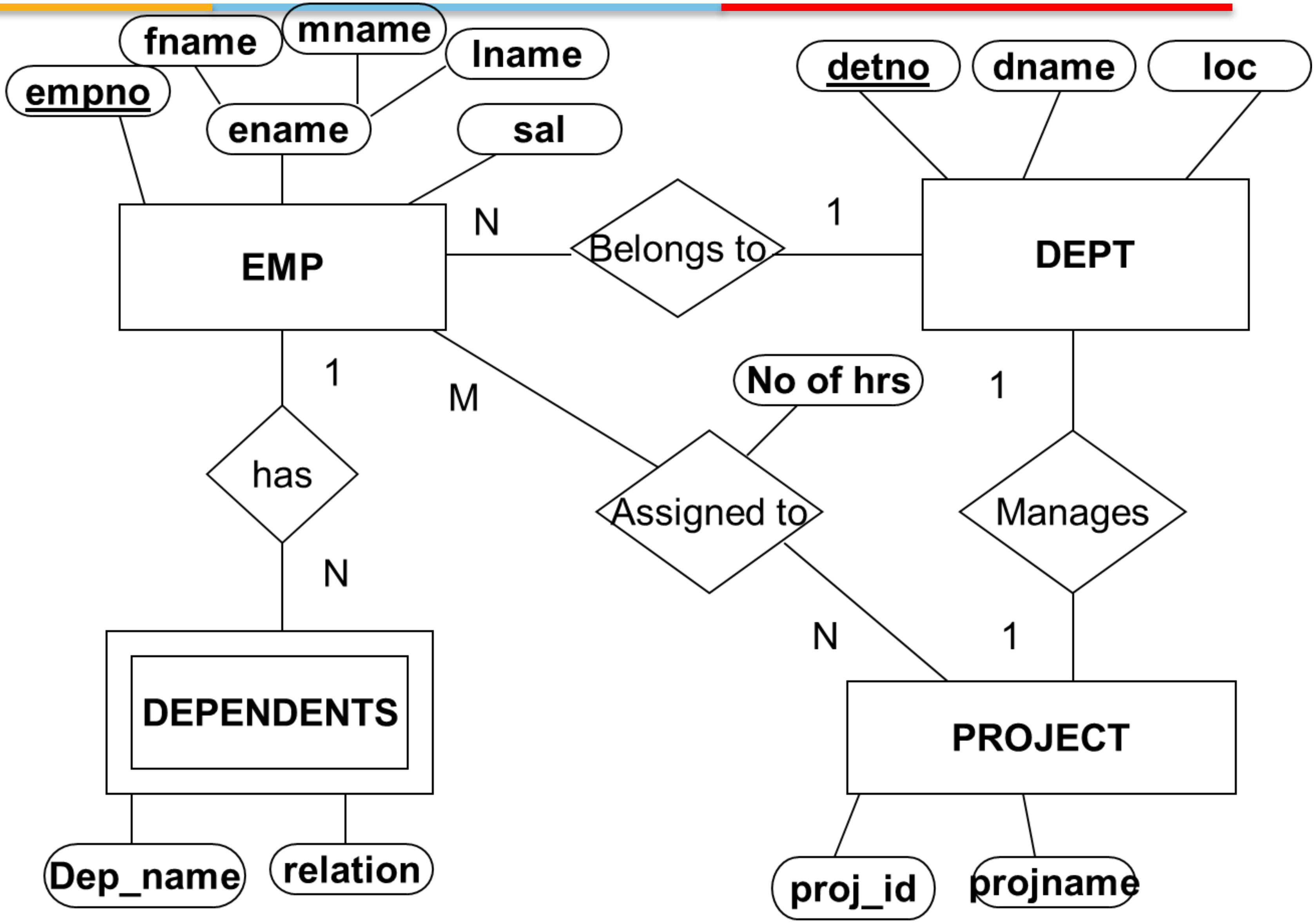
Cardinality Ratio/ Mapping Cardinality

- Mapping Cardinality express the number of entities to which another entity can be associated via a relationship set.
- Once the entities & relationship between the entities is identified you can represent them in a ER diagram



- 1-1
  Supplier------Key Person
- 1-M
  Dept-----Emp
- M-1
  Emp-----Dept
- M:M
  Student----Subject

ER Example


fname
mname
lname
empno
ename
sal
detno
dname
loc
EMP
N
Belongs to
1
DEPT
1
M
No of hrs
1
has
Assigned to
Manages
N
N
1
DEPENDENTS
PROJECT
Dep_name
relation
proj_id
projname

ER …(contd.)

This schema has 3 entity types

- EMPLOYEE
- DEPT
- PROJECT

& one weak entity type DEPENDENT

There is M:N relationship between PROJECT and EMP entity sets

There is a 1:1 relationship between DEPT and PROJECT

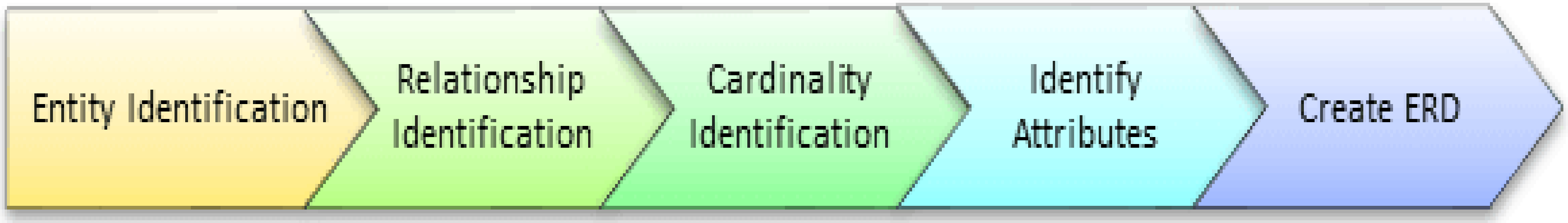
There is a 1:M relationship between DEPT and EMP entity sets

No of hrs is an attribute on the relation “assigned to” between EMP and PROJECT

Entity Relationship Model

Steps to Create an ERD

Following are the steps to create an ERD.



Entity Identification

Relationship Identification

Cardinality Identification

Identify Attributes

Create ERD

Let's study them with an example:

In a university, a Student enrols in Courses.

A student must be assigned to at least one or more Courses.

Each course is taught by a single Professor.

To maintain instruction quality, a Professor can deliver only one course

Entity Relationship Model

Step 1) Entity Identification – Identification of entities in the given application

We have three entities

Student, Course and Professor and are represented as follows

Student **Course** **Professor**

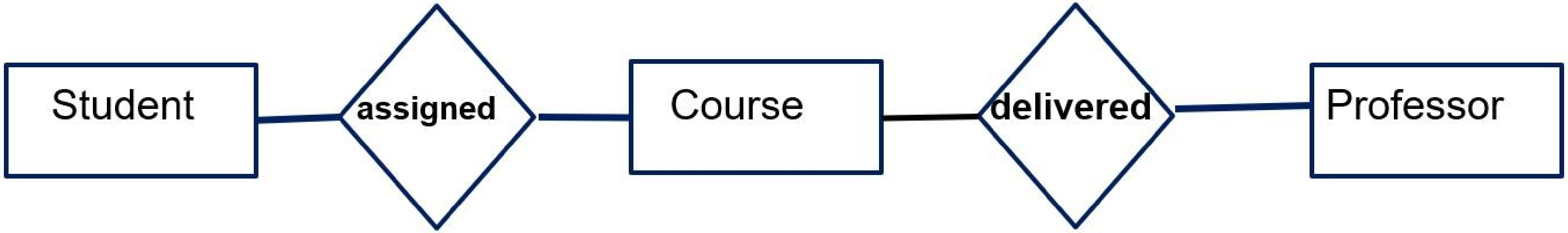
Entity Relationship Model

Step 2) Relationship Identification – Identifying the relationships between the entities

We have the following two relationships

1. The student is **assigned** a course (Relationship between a Student and the course)

2. Professor **delivers** a course (Relationship between a Course and the Professor)



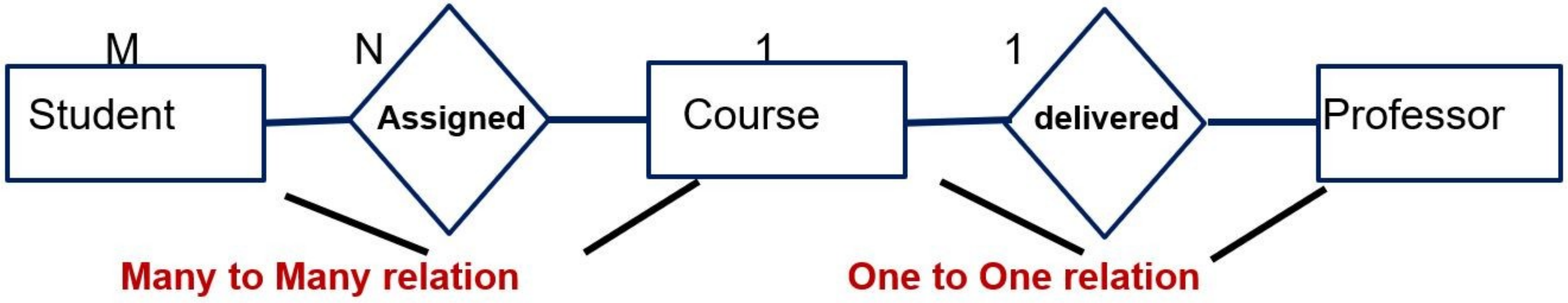
Entity Relationship Model

Step 3) Cardinality Identification – Identification of mapping cardinalities between the entities.

A student can be assigned **multiple** courses

A Professor can deliver only **one** course

A Course can be taken by only **one** Professor


M
N
1
1
Student
Assigned
Course
delivered
Professor
Many to Many relation
One to One relation

Entity Relationship Model

Step 4) Identify Attributes – Identifying the attributes of each entity of the application and the primary key of the entity set.

For Entity Set Student, the attributes are Name and Number in which Number is the Primary key

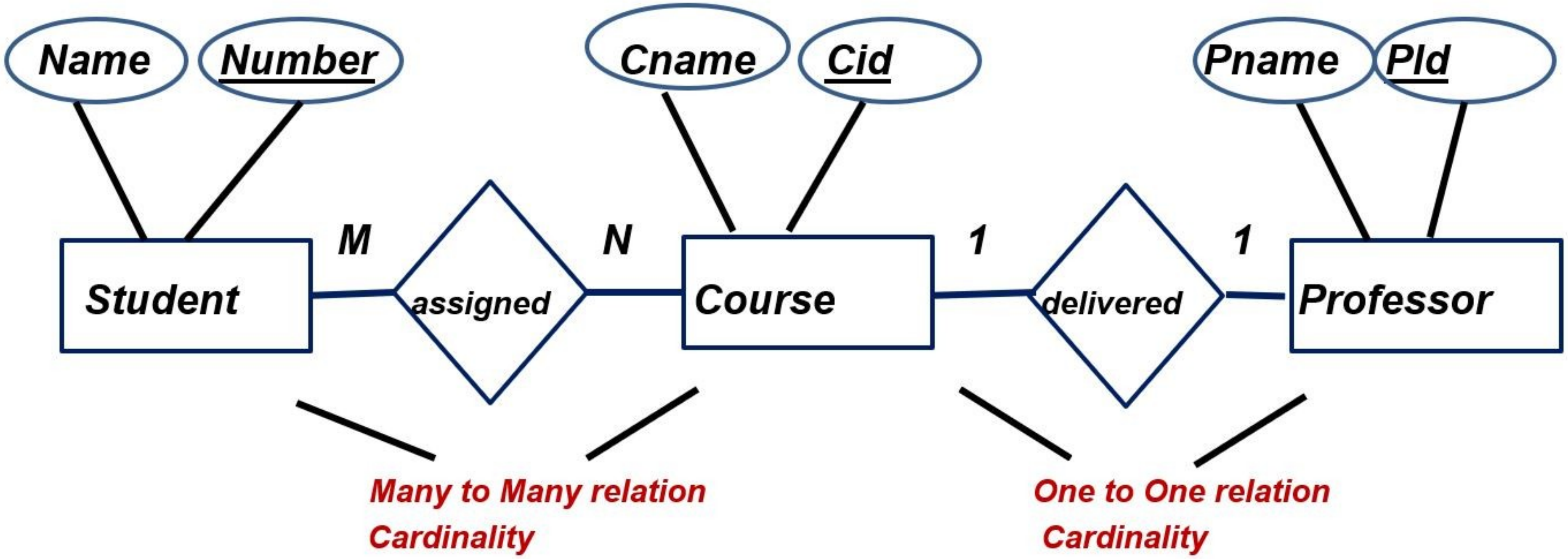
For Entity Set Course, the attributes are Course Name and Course Id in which Course Id is the Primary key

For Entity Set Professor, the attributes are PName and PNumber in which PNumber is the Primary key

Entity Relationship Model

Step 5) Create the ERD – Produce the Entity relationship diagram by linking all the attributes with the corresponding Entities.

The resultant ERD for the given application is


Name
Number
Cname
Cid
Pname
PId
Student
M
assigned
N
Course
1
delivered
1
Professor
Many to Many relation Cardinality
One to One relation Cardinality

Enhanced features of ER Model

Enhanced-ER (EER) Model Concepts

- Includes all modeling concepts of basic ER
- Additional concepts: subclasses / super classes, specialization/generalization, categories, attribute inheritance
- The resulting model is called the enhanced-ER or Extended ER (E2R or EER) model
- It is used to model applications more completely and accurately if needed
- It includes some object-oriented concepts, such as inheritance

Subclasses and Super classes

- An entity type may have additional meaningful sub groupings of its entities
- Example: EMPLOYEE may be further grouped into SECRETARY, ENGINEER, MANAGER, TECHNICIAN, SALARIED_EMPLOYEE, HOURLY_EMPLOYEE,…
- Each of these groupings is a subset of EMPLOYEE entities
- Each is called a subclass of EMPLOYEE
- EMPLOYEE is the superclass for each of these subclasses
- These are called superclass/subclass relationships.
- Example: EMPLOYEE/SECRETARY, EMPLOYEE/TECHNICIAN
- These are also called IS-A relationships (SECRETARY IS-A EMPLOYEE, TECHNICIAN IS-A EMPLOYEE, …)

Attribute Inheritance in Superclass / Subclass Relationships

- An entity that is member of a subclass *inherits* all attributes of the entity as a member of the superclass
- It also inherits all relationships.

Specialization

Specialization is the process of defining a set of subclasses of a superclass

The set of subclasses is based upon some distinguishing characteristics of the entities in the superclass

Example: {SECRETARY, ENGINEER, TECHNICIAN} is a specialization of EMPLOYEE based upon *job type*.

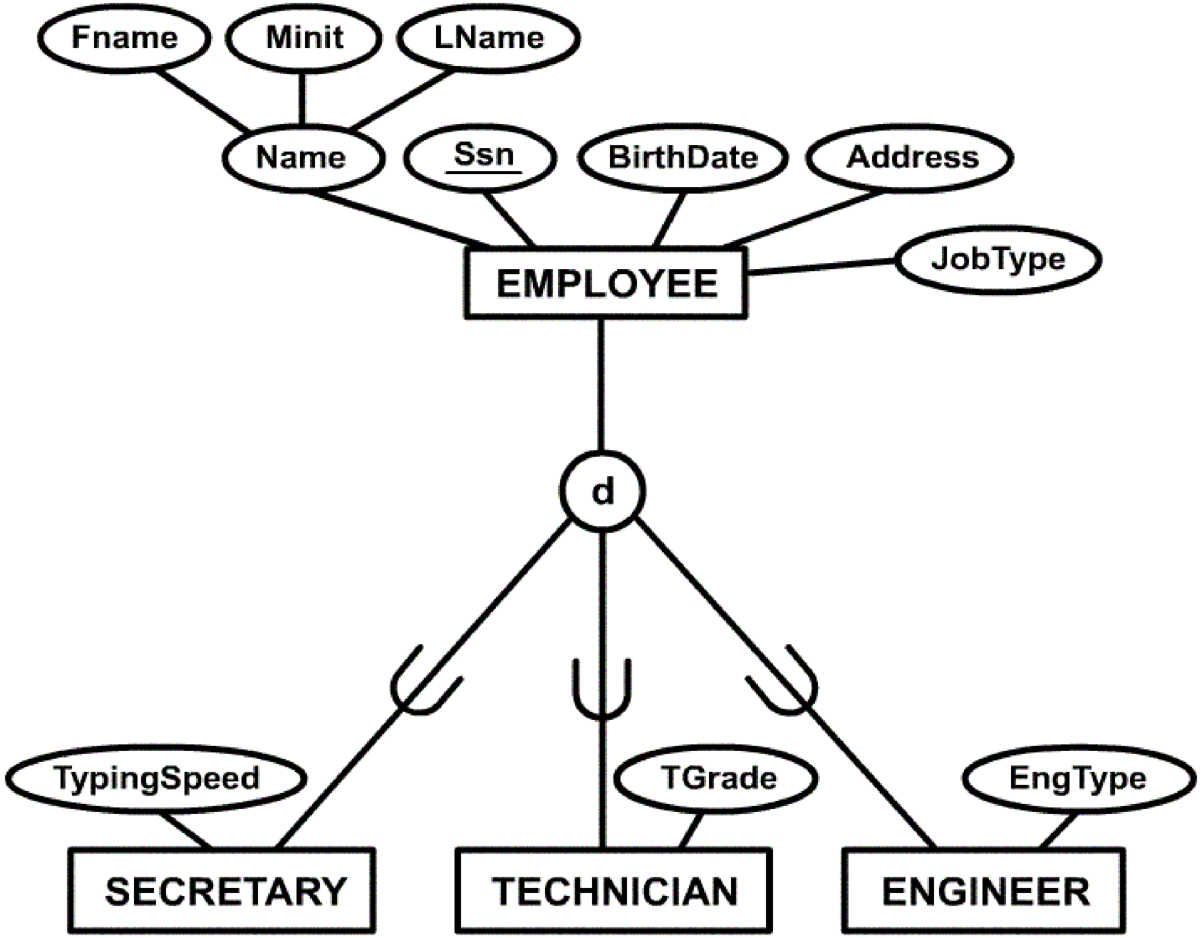
- May have several specializations of the same superclass

Specialization

Example: Another specialization of EMPLOYEE based on *method of pay* is {SALARIED_EMPLOYEE, HOURLY_EMPLOYEE}.

- Superclass/subclass relationships and specialization can be diagrammatically represented in EER diagrams
- Attributes of a subclass are called specific attributes. For example, TypingSpeed of SECRETARY
- The subclass can participate in specific relationship types. For example, BELONGS_TO of HOURLY_EMPLOYEE

Example of a Specialization


Fname
Minit
LName
Name
Ssn
BirthDate
Address
JobType
EMPLOYEE
d
TypingSpeed
TGrade
EngType
SECRETARY
TECHNICIAN
ENGINEER

Generalization

- The reverse of the specialization process
- Several classes with common features are generalized into a superclass; original classes become its subclasses
- Example: CAR, TRUCK generalized into VEHICLE; both CAR, TRUCK become subclasses of the superclass VEHICLE.
  - We can view {CAR, TRUCK} as a specialization of VEHICLE
  - Alternatively, we can view VEHICLE as a generalization of CAR and TRUCK

Constraints on Specialization and Generalization

- If we can determine exactly those entities that will become members of each subclass by a condition, the subclasses are called ***predicate-defined*** (or condition-defined) subclasses
  - Condition is a constraint that determines subclass members
  - Display a predicate-defined subclass by writing the predicate condition next to the line attaching the subclass to its superclass.
  - Two types of conditions or constraints
    - Attribute defined
    - User defined

Constraints on Specialization and Generalization

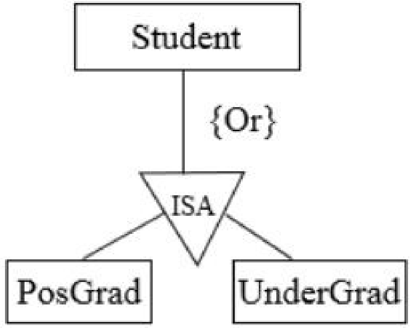
- If all subclasses in a specialization have membership condition on same attribute of the superclass, such specialization is called an ***attribute defined*-specialization**
  - Attribute is called the defining attribute of the specialization
  - Example: JobType is the defining attribute of the specialization {SECRETARY, TECHNICIAN, ENGINEER} of EMPLOYEE
- If no condition determines membership, the subclass is called ***user-defined***
  - Membership in a subclass is determined by the database users by applying an operation to add an entity to the subclass
  - Membership in the subclass is specified individually for each entity in the superclass by the user. Eg. Dependent of an employee can be wife, parent, child

Constraints on Specialization and Generalization

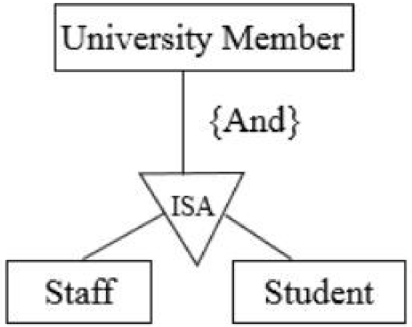
Two other conditions apply to a specialization/generalization:

1. Disjoint Constraint:

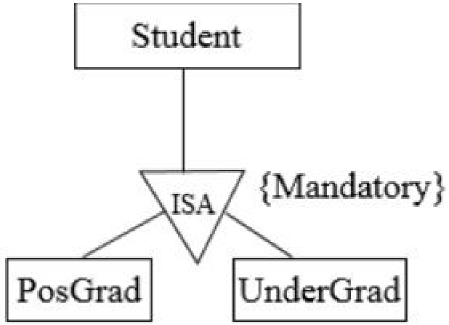
- Specifies that the subclasses of the specialization must be **disjoint** be a member of at most one of the subclasses of the specialization)


Student
{Or}
ISA
PosGrad
UnderGrad

- Specified by d in EER diagram
- If not disjointed, **overlapping constraint**; that is the same entity may be a member of more than one subclass of the specialization
- Specified by o in EER diagram

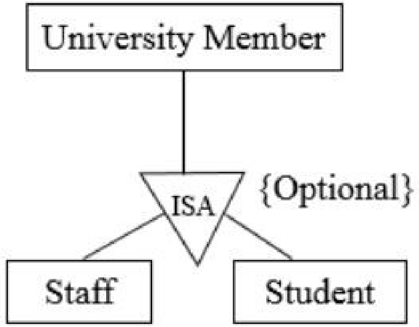

University Member
{And}
ISA
Staff
Student

Constraints on Specialization and Generalization


Student
ISA
{Mandatory}
PosGrad
UnderGrad

2. Completeness Constraint:

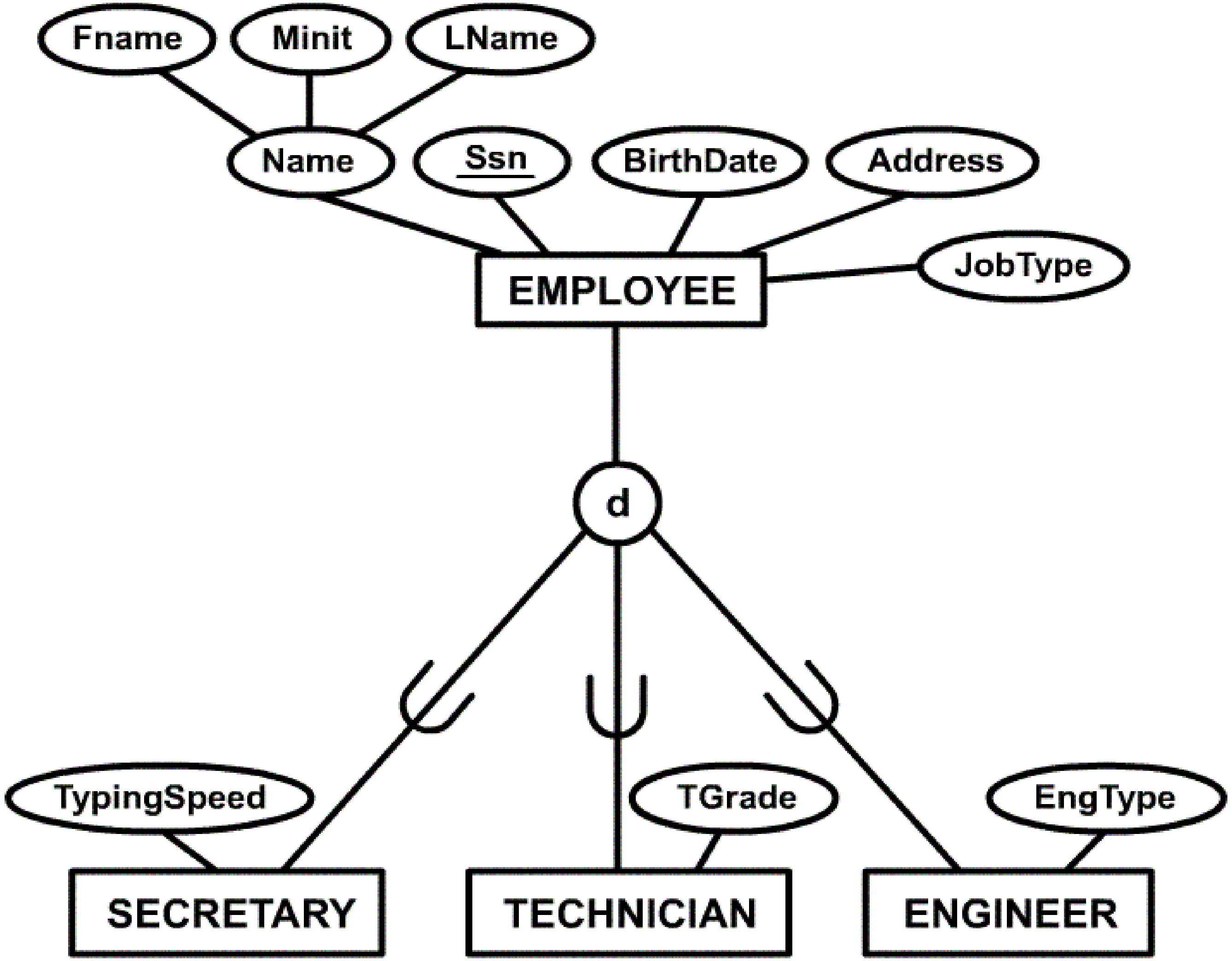
- **Total** specifies that every entity in the superclass must be a member of some subclass in the specialization/ generalization
- Shown in EER diagrams by a double line
- **Partial** participation - Not all entities are involved in the relationship.
- Partial participation is represented by single lines.


University Member
ISA
{Optional}
Staff
Student

Constraints on Specialization and Generalization

- Hence, we have four types of specialization/generalization:
    - Disjoint, total
    - Disjoint, partial
    - Overlapping, total
    - Overlapping, partial
- Note: Generalization usually is total because the superclass is derived from the subclasses.

Example of disjoint partial Specialization


Fname
Minit
LName
Name
Ssn
BirthDate
Address
JobType
EMPLOYEE
d
TypingSpeed
TGrade
EngType
SECRETARY
TECHNICIAN
ENGINEER

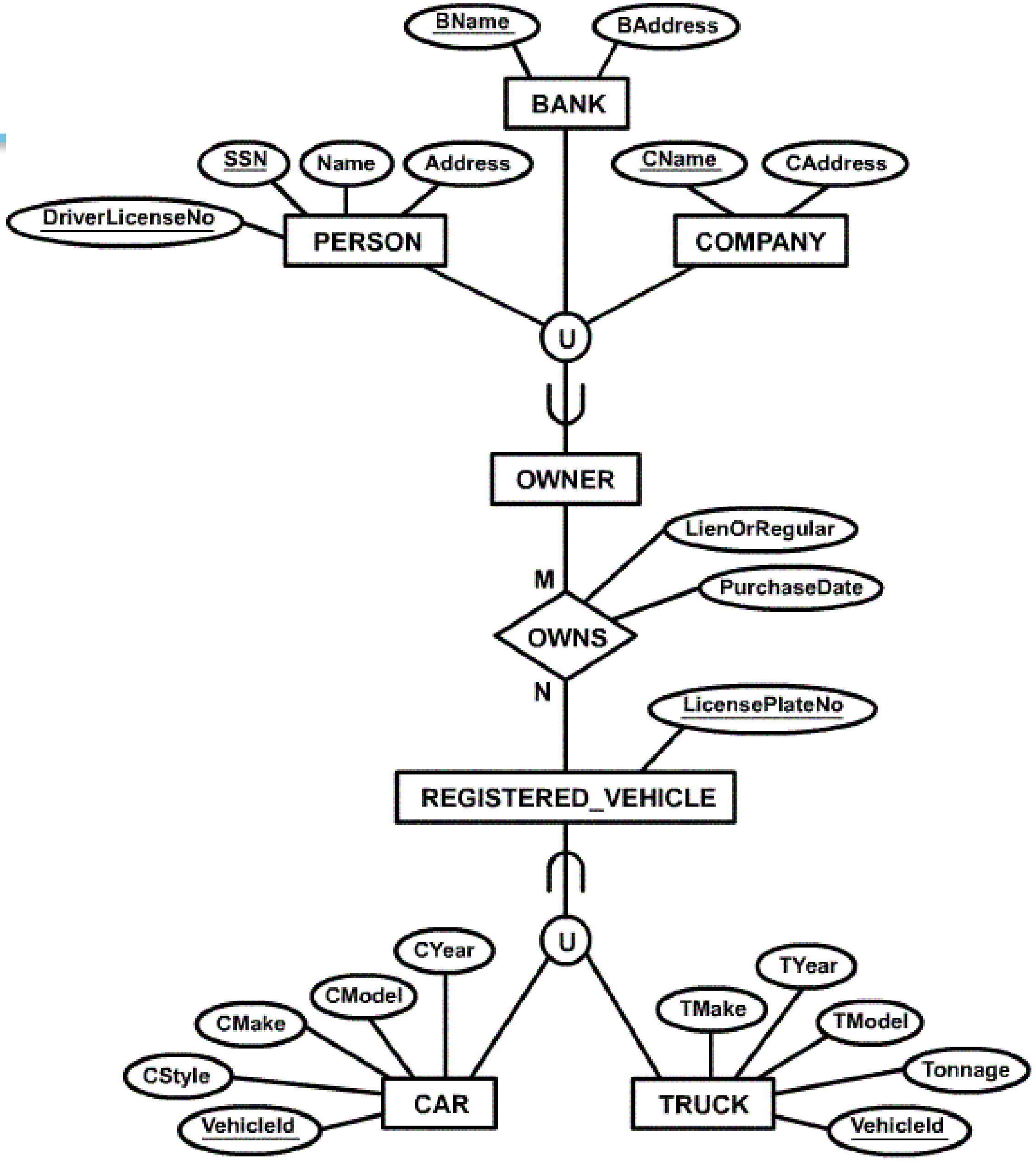
Categories (UNION TYPES – Multiple Inheritance)

- All of the superclass/subclass relationships we have seen thus far have a single superclass
- A shared subclass is subclass in more than one distinct superclass/subclass relationships, where each relationships has a single superclass (multiple inheritance)
- In some cases, need to model a single superclass/subclass relationship with **more than one superclass**
- Super classes represent different entity types
- Such a subclass is called a category or UNION TYPE

Categories (UNION TYPES)

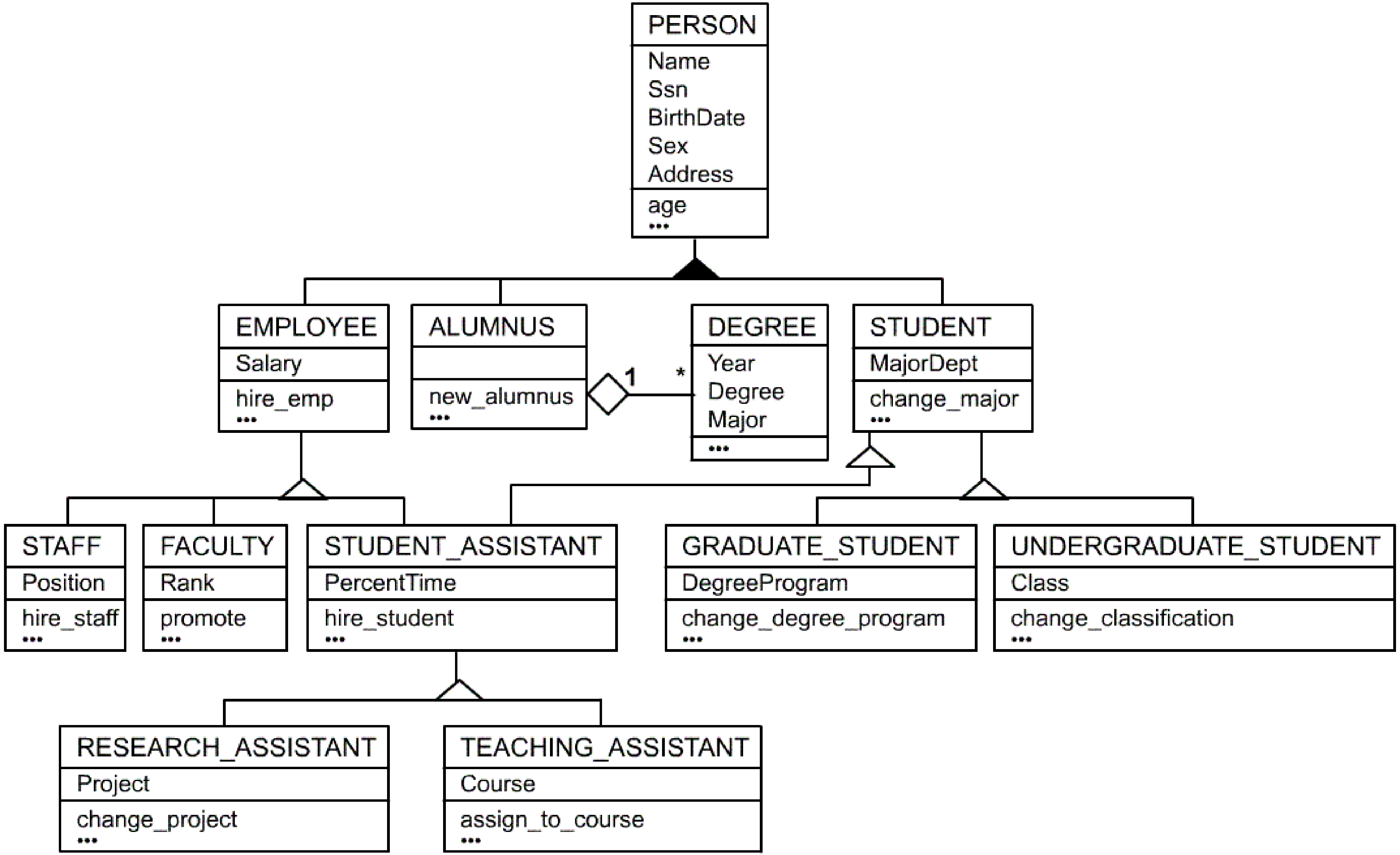
- Example: Database for vehicle registration, vehicle owner can be a person, a bank (holding a lien on a vehicle) or a company.
- Category (subclass) OWNER is a subset of the union of the three super classes COMPANY, BANK, and PERSON
- A category member must exist in at least one of its super classes

Example of categories (UNION TYPES)



BName
BAddress
BANK
SSN
Name
Address
CName
CAddress
DriverLicenseNo
PERSON
COMPANY
U
OWNER
LienOrRegular
M
PurchaseDate
OWNS
N
LicensePlateNo
REGISTERED_VEHICLE
U
CYear
CModel
CMake
CStyle
VehicleId
CAR
TYear
TMake
TModel
Tonnage
VehicleId
TRUCK

UML Example for Displaying Specialization / Generalization


PERSON
Name
Ssn
BirthDate
Sex
Address
age
•••
EMPLOYEE
Salary
hire_emp
•••
ALUMNUS
new_alumnus
•••
1
*
DEGREE
Year
Degree
Major
•••
STUDENT
MajorDept
change_major
•••
STAFF
Position
hire_staff
•••
FACULTY
Rank
promote
•••
STUDENT_ASSISTANT
PercentTime
hire_student
•••
GRADUATE_STUDENT
DegreeProgram
change_degree_program
•••
UNDERGRADUATE_STUDENT
Class
change_classification
•••
RESEARCH_ASSISTANT
Project
change_project
•••
TEACHING_ASSISTANT
Course
assign_to_course
•••

Thank
you

Database Design

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Contact Session 3: Relational Model

Session Outline

- Relational Model Concepts
- Relational Model Constraints and Relational Database Schemas
- Update Operations and Dealing with Constraint Violations

Copyright @2007 Ramez Elmasri, Shamkant B. Navathe

Recap-QUIZ

Which type of entity cannot exist in the database unless another type of entity also exists in the database?

A. Weak entity
B. Strong entity
C. Dependent entity
D. Independent entity

Answer: A – Weak Entity - An entity that depends on some other entity for its existence

Recap-QUIZ

The attribute name could be structured as an attribute consisting of first name, middle initial, and last name. This type of attribute is called

a) Simple attribute

b) Composite attribute

c) Multivalued attribute

d) Derived attribute

Answer: B - Composite attribute

Recap-QUIZ

______________ express the number of entities to which another entity can be associated via a relationship set.

a) Mapping Cardinality

b) Relational Cardinality

c) Participation Constraints

d) None of the mentioned

Answer: A – Mapping cardinality - Mapping cardinality is **the maximum number of relationship instances in which an entity can participate**

Recap-QUIZ

The entity which is created through specialization is considered as

a) Medium Class of entity

b) Mini Class of entity

c) Sub class of entity

D) Super class of entity

Answer: Sub class of entity

Recap-QUIZ

- Generalization is

ANS: Grouping entity types, bottom up

Recap-Exercise

We wish to create a database for **BITS Data Science trainings.** For this, we must store data about the Participants and the Faculty. For each course participant, identified by an ID, we want to store her social security number, surname, age, sex, place of birth, employer’s name, address and telephone number, previous employers (and periods employed), the courses attended and the final assessment for each course. If a participant is self-employed, we need to know her area of expertise, and, if appropriate, her title. For somebody who works for a company, we store the level and position held. Each course has a code and a title and any course can be given any number of times. Each time a particular course is given, we will call it an ‘instance’ of the course. For each instance, we represent the start date, the end date, and the number of participants. For each Faculty, we will show the SSN, surname, age, place of birth, the instance of the course currently being taught, those taught in the past and the courses that the faculty is qualified to teach. All the faculty’s telephone numbers are also stored. An instructor can be permanently employed by BITS or can be a guest faculty.

1) Identify the entity sets.
2) Identify the relationships .
3) Identify the structural constraints of all the relationships
4) Identify the specializations /generalizations involved.

Relational Model Concepts

- The relational Model of Data is based on the concept of a *Relation*
- A Relation is a mathematical concept based on the ideas of sets
- The model was first proposed by Dr. E.F. Codd of IBM Research in 1970

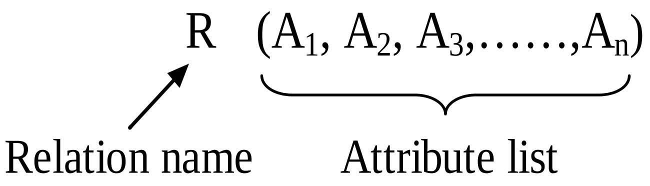
Copyright @2007 Ramez Elmasri, Shamkant B. Navathe

Relational Model Characteristics

- Using this representational (or implementation) model we represent a database as collection of relations.
- The notion of relation here is different from the notion of relationship used in ER modeling.
- Relation is the main construct for representing data in relational model.
- Every relation consists of a relation schema and Relation instance.

Relational Model Characteristics

Relation Schema is denoted by



$R\ \underbrace{(A_1, A_2, A_3,\ldots\ldots,A_n)}_{\text{Attribute list}}$

Relation name

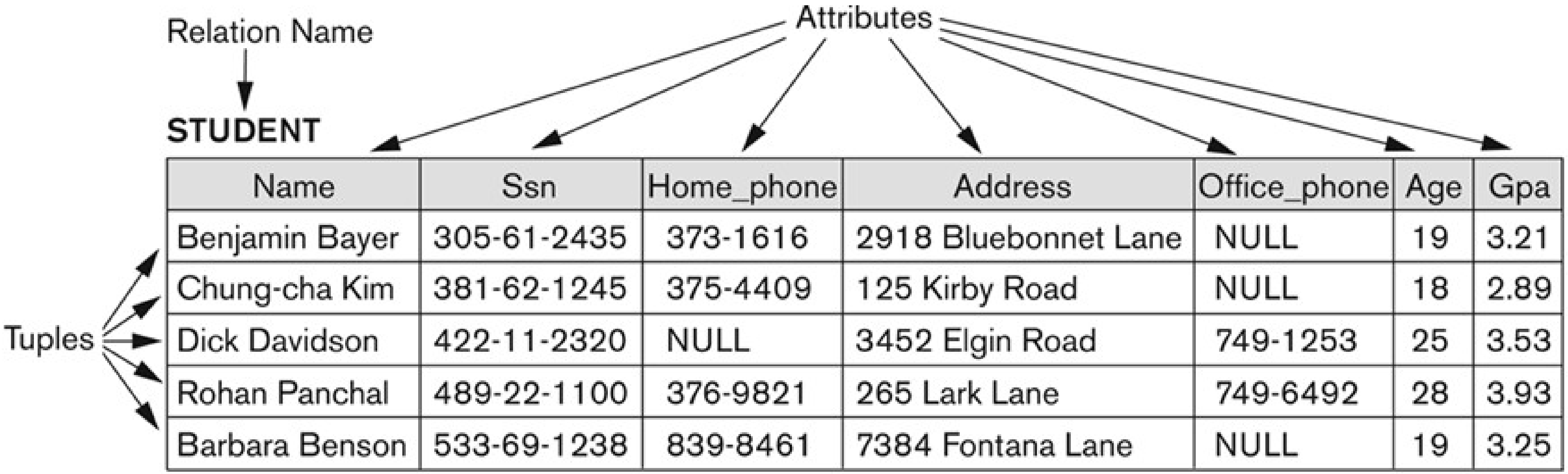
- The number of columns in a relation is known as its ***degree* or *arity***'.

- A relation of degree seven, which stores information about university students, would contain seven attributes describing each student. as follows:

  - STUDENT(Name, Ssn, Home_phone, Address, Office_phone, Age, Gpa)

- Using the data type of each attribute, the definition is sometimes written as:
  - STUDENT(Name: string, Ssn: string, Home_phone: string, Address: string,Office_phone: string, Age: integer, Gpa: real)

Relational Model Characteristics


Relation Name
STUDENT
Attributes
Tuples

Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	381-62-1245	375-4409	125 Kirby Road	NULL	18	2.89
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	749-1253	25	3.53
Rohan Panchal	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	NULL	19	3.25

Relation instance

Relation instance or Relation State (r) of R (thought of as a table)

- Each *row* in the table represents a collection of related data.
- Each row contains facts about some entity of same entity-set.

$R = (A_1, A_2, A_3, \ldots\ldots, A_n)$

$r(R)$ is a set of n tuples in R

$r = \{t_1, t_2, t_3, \ldots\ldots, t_n\}$

- ***r*** is an **instance** of R each ***t*** is a tuple and is an ordered list of **values**.
- $t = (v_1, v_2, \ldots, v_n)$ where v_i is an element of domain of A_i

- Attribute A of relation R is accessed by notation - R.A.

  Ex: Student (name, age, branch). Here Student is the relation name.

  Student.age - denotes age attribute of Student relation.

Relational Model Characteristics

- Entities of each type/set are stored as rows in a single relation.
- Hence in general, a ***relation*** corresponds to a single entity type in ER diagram.
- In some cases a **relationship** between two entities can have some specific **attributes** which can be captured in a relation (table).
- A row is called a ***tuple***.
- The columns of the table represent attributes of that entity type.
- The column header is known as ***attribute*** or ***field***.
- ***Data type or format*** of an attribute: is the format of data for that attribute. Ex. Character strings, numeric, alphanumeric etc.
- Values that can appear in a column for any row is called the ***domain*** of that attribute.
    - Example: “USA_phone_numbers” are the set of 10 digit phone numbers valid in the U.S.

Relational Model Characteristics

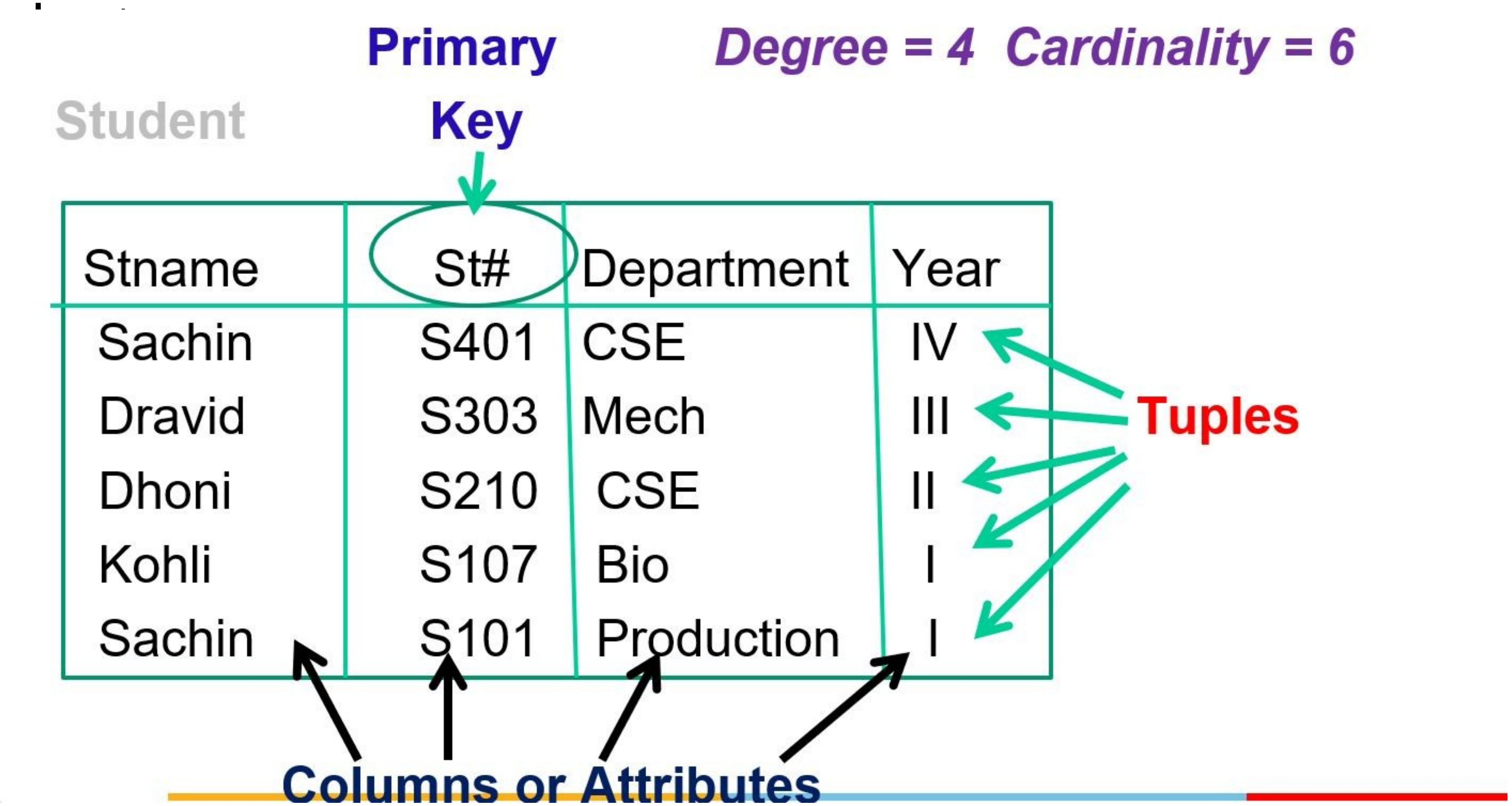
Relational Database Schema is denoted by

$$S = \underbrace{\{R_1, R_2, \ldots\ldots, R_n\}}_{\text{Relations in the database (tables)}}$$

Database name

Relational Model

For example the following is the relation for the Stu


Primary Key
Degree = 4 Cardinality = 6
Student
Tuples
Columns or Attributes

Stname	St#	Department	Year
Sachin	S401	CSE	IV
Dravid	S303	Mech	III
Dhoni	S210	CSE	II
Kohli	S107	Bio	I
Sachin	S101	Production	I

Characteristics of a Relation

- Ordering of tuples is not significant.
- Ordering of values in a tuple is important.
- Values in a tuple under each column must be atomic (simple & single).
  - Example Multiple Phone numbers to be stored separately under different columns namely phone #1, phone#2, etc.

- NULL values : used to represent the values of attributes that may be unknown or may not apply to a tuple
- Interpretation (Meaning) of a Relation : a type of assertion
  - An assertion is a statement in SQL that ensures a certain condition will always exist in the database. Assertions are like column and table constraints.

Relational Model Characteristics

- **Key of a Relation**:
  - Each row has a value of a data item (or set of items) that uniquely identifies that row in the table
    - Called the *key*
  - In the STUDENT table, SSN is the key

Copyright @2007 Ramez Elmasri, Shamkant B. Navathe

Relational Model - Keys

Key – An attribute in a relation that is used to distinguish one tuple from another tuple is referred as the Key.

Different types of keys

1. **Primary and Secondary keys**

If the key uniquely identifies only one tuple/ record from all tuples / record is called as the **Primary Key**.

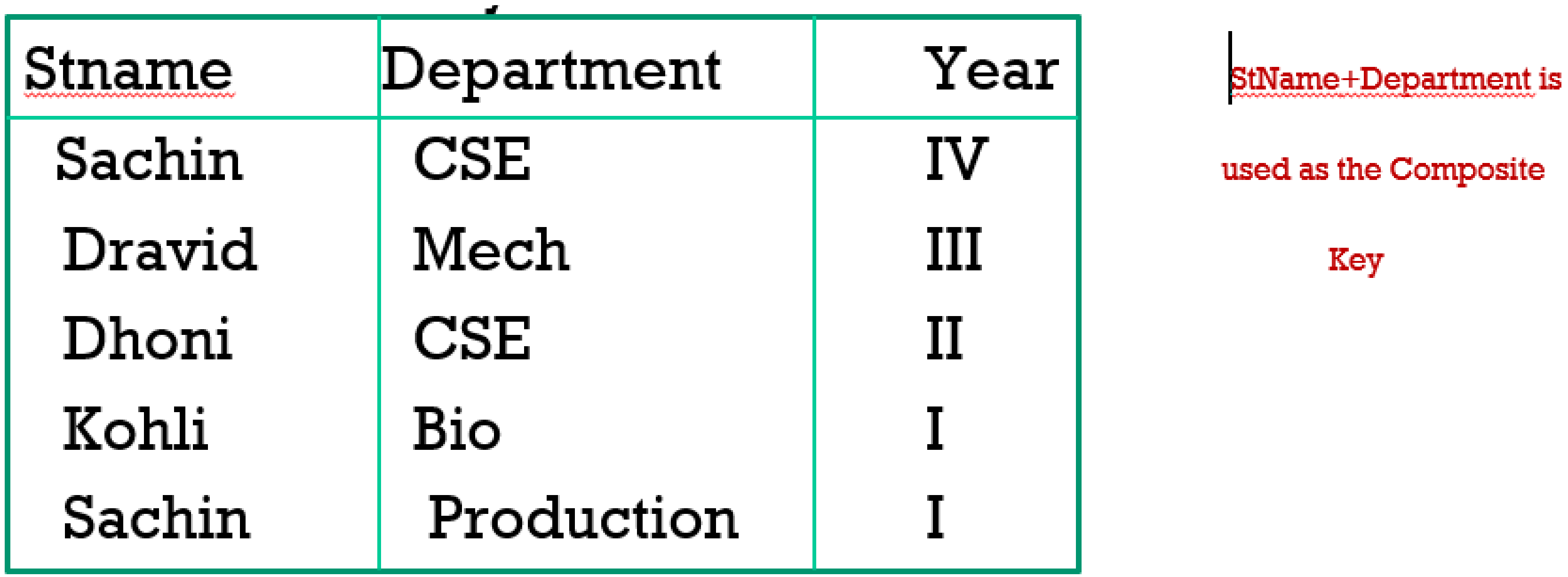
Otherwise the key is called as the **Secondary key**.

Example: In the above relation **Student**, the attributes Stu# and StName are the Primary and Secondary Keys respectively.

Relational Model - keys

2. **Composite or concatenated key**

When a table has no primary key for unique identification, we can combine more than one secondary keys to identify unique record/tuple, the key is called as the Composite or Concatenated keys.



Stname	Department	Year
Sachin	CSE	IV
Dravid	Mech	III
Dhoni	CSE	II
Kohli	Bio	I
Sachin	Production	I

StName+Department is

used as the Composite

Key

Relational Model - keys

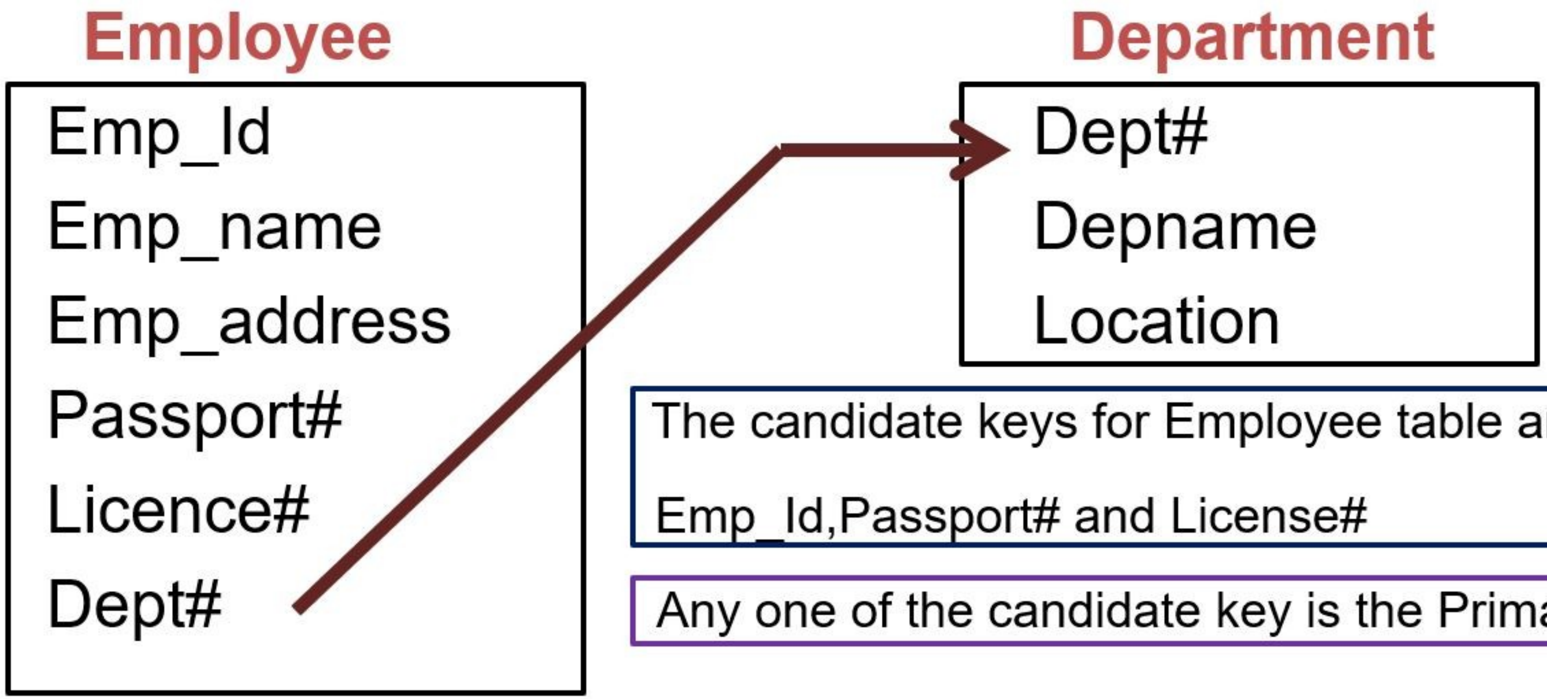
3. Candidate Key – It is an attribute or a set of attributes that can act as a primary key which can uniquely identify the record/tuple in a relation

4. Alternate key – If a relation has two separate primary key attributes say a1 and a2 and one of the attribute is used as the primary key, the other is called as alternate key for the primary key.

5. Foreign Key – It is one of the attribute in a table which is a primary key of another table.

The candidate, alternate and foreign keys are explained in the following tables

Relational Model - keys



Employee

Emp_Id
Emp_name
Emp_address
Passport#
Licence#
Dept#

Department

Dept#
Depname
Location

The candidate keys for Employee table are

Emp_Id,Passport# and License#

Any one of the candidate key is the Primary Key

Dep# is the foreign Key

Passport# or Licence# is the alternate Key for Emp_Id

Relational Model - Rules

Best Practices for creating a Relational Model

- Data need to be represented as a collection of relations
- Each relation should be depicted clearly in the table
- Rows should contain data about instances of an entity
- Columns must contain data about attributes of the entity
- Cells of the table should hold a single value
- Each column should be given a unique name
- No two rows can be identical
- The values of an attribute should be from the same domain

Terminology

Informal Terms	**Formal Terms**
Table	Relation
Column Header	Attribute
All possible Column Values	Domain
Row	Tuple
Table Definition	Schema of a Relation
Populated Table	State of the Relation

Relational Constraints

Constraints are restrictions on data of a relation.

- *Domain level Constrains* – Format of data Ex. Character numeric etc.
- *Entity integrity constraints* – Primary key, unique key
- *Referential integrity* constraints– Foreign key
- Semantic Constraints.

Relational Integrity Constraints

- Constraints are **conditions** that must hold on **all** valid relation states.
- There are three *main types* of constraints in the relational model:
  - **Key** constraints
  - **Entity integrity** constraints
  - **Referential integrity** constraints
- Another implicit constraint is the **domain** constraint
  - Every value in a tuple must be from the *domain of its attribute* (or it could be **null**, if allowed for that attribute)

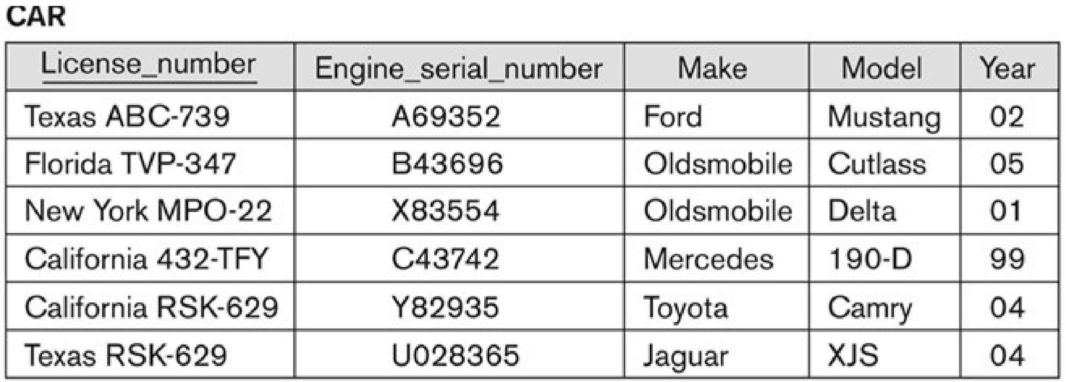
Copyright @2007 Ramez Elmasri, Shamkant B. Navathe

Key Constraints

- **Superkey** of R:
  - Is a set of attributes SK of R with the following condition:
    - No two tuples in any valid relation state r(R) will have the same value for SK
    - That is, for any distinct tuples t1 and t2 in r(R), t1[SK] $\neq$ t2[SK]
    - This condition must hold in *any valid state* r(R)
- **Key** of R:
- It is used to uniquely identify any record or row of data from the table.
    - ID is used as a key in the Student table because it is unique for each student. In the PERSON table, passport_number, license_number, SSN are keys since they are unique for each person.
- It is also used to establish and identify relationships between tables.
  - A key is a "minimal" superkey
  - That is, a key is a superkey K such that removal of any attribute from K results in a set of attributes that is not a superkey (does not possess the superkey uniqueness property)

Key Constraints (continued)

- Example: Consider the CAR relation scher
  - CAR(State, Reg#, SerialNo, Make, Model, Y
  - CAR has two keys:
    - Key1 = {State, Reg#}
    - Key2 = {SerialNo}
  - Both are also superkeys of CAR
  - {SerialNo, Make} is a superkey but *not* a key.
- In general:
  - Any *key* is a *superkey* (but not vice versa)
  - Any set of attributes that *includes a key* is a *superkey*
  - A *minimal* superkey is also a key.



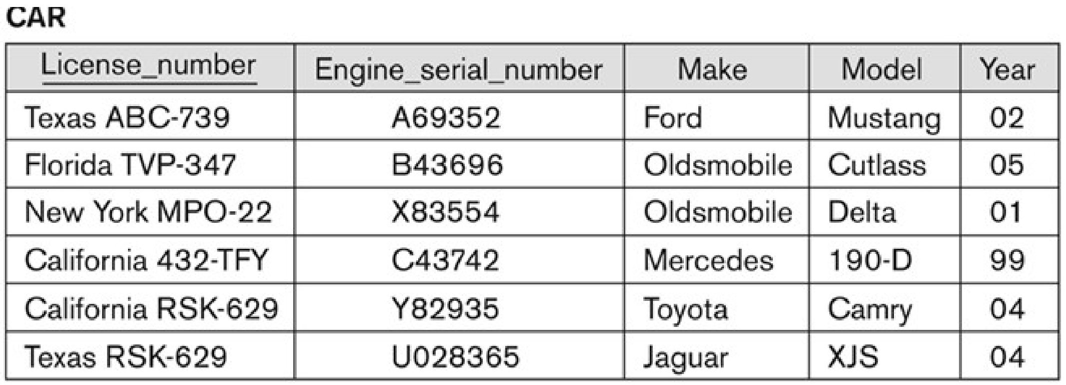
CAR

License_number	Engine_serial_number	Make	Model	Year
Texas ABC-739	A69352	Ford	Mustang	02
Florida TVP-347	B43696	Oldsmobile	Cutlass	05
New York MPO-22	X83554	Oldsmobile	Delta	01
California 432-TFY	C43742	Mercedes	190-D	99
California RSK-629	Y82935	Toyota	Camry	04
Texas RSK-629	U028365	Jaguar	XJS	04

» Further Reading: https://www.javatpoint.com/dbms-keys#:~:text=Super%20key%20is%20an%20attribute,can%20also%20be%20a%20key.

Key Constraints (continued)

- If a relation has several **candidate keys**, one is chosen arbitrarily to be the **primary key**.
  - The primary key attributes are underlined
- Example: Consider the CAR relation scher
  - CAR(State, Reg#, SerialNo, Make, Model, Y
  - We chose SerialNo as the primary key



CAR

License_number	Engine_serial_number	Make	Model	Year
Texas ABC-739	A69352	Ford	Mustang	02
Florida TVP-347	B43696	Oldsmobile	Cutlass	05
New York MPO-22	X83554	Oldsmobile	Delta	01
California 432-TFY	C43742	Mercedes	190-D	99
California RSK-629	Y82935	Toyota	Camry	04
Texas RSK-629	U028365	Jaguar	XJS	04

- The primary key value is used to *uniquely identify* each tuple in a relation
  - Provides the tuple identity
- Also used to *reference* the tuple from another tuple
  - General rule: Choose as primary key the **smallest** of the candidate keys (in terms of size)
  - Not always applicable – choice is sometimes subjective

Exercise-1

Consider the relation

CLASS(Course#, Univ_Section#, Instructor_name, Semester, Building_code, Room#, Time_period, Weekdays, Credit_hours).

This represents classes taught in a university, with unique Univ_section#s. Identify what you think should be various candidate keys with justification.

Entity Integrity

- **Entity Integrity:**
  - The *primary key attributes* PK of each relation schema R in S cannot have null values in any tuple of r(R).
    - This is because primary key values are used to *identify* the individual tuples.
    - t[PK] ≠ null for any tuple t in r(R)
    - If PK has several attributes, null is not allowed in any of these attributes
  - Note: Other attributes of R may be constrained to disallow null values, even though they are not members of the primary key.

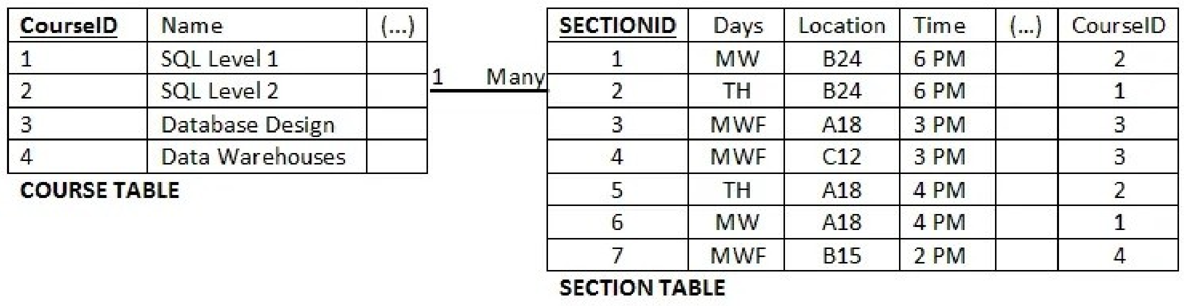
Referential Integrity

- A constraint involving **two** relations
- Used to specify a **relationship** among tuples in two relations:
  - The **referencing relation** and the **referenced relation**.

Referential Integrity

- Tuples in the **referencing relation** R1 have attributes FK (called **foreign key** attributes) that reference the primary key attributes PK of the **referenced relation** R2.
  - A tuple t1 in R1 is said to **reference** a tuple t2 in R2 if t1[FK] = t2[PK].
- A referential integrity constraint can be displayed in a relational database schema as a directed arc from R1.FK to R2.

Referential Integrity



CourseID	Name	(...)
1	SQL Level 1	
2	SQL Level 2	
3	Database Design	
4	Data Warehouses	

COURSE TABLE

SECTIONID	Days	Location	Time	(...)	CourseID
1	MW	B24	6 PM		2
2	TH	B24	6 PM		1
3	MWF	A18	3 PM		3
4	MWF	C12	3 PM		3
5	TH	A18	4 PM		2
6	MW	A18	4 PM		1
7	MWF	B15	2 PM		4

SECTION TABLE

Example from a Course/Class database.

- Course(**CrsCode**, DeptCode, Description)
- Class(**CrsCode, Section**, ClassTime)

- The referential integrity constraint states that CrsCode in the Class table must match a valid CrsCode in the Course table.
- In this situation, it's not enough that the CrsCode and Section in the Class table make up the PK, we must also enforce referential integrity.
- When setting up referential integrity it is important that the PK and FK have the same data types and come from the same domain, otherwise the relational database management system (RDBMS) will not allow the join.

Referential Integrity (or foreign key) Constraint

- Statement of the constraint
  - The value in the foreign key column (or columns) FK of the **referencing relation** R1 can be **either**:
    - (1) a value of an existing primary key value of a corresponding primary key PK in the **referenced relation** R2, or
    - (2) a **null**.
- In case (2), the FK in R1 should **not** be a part of its own primary key.

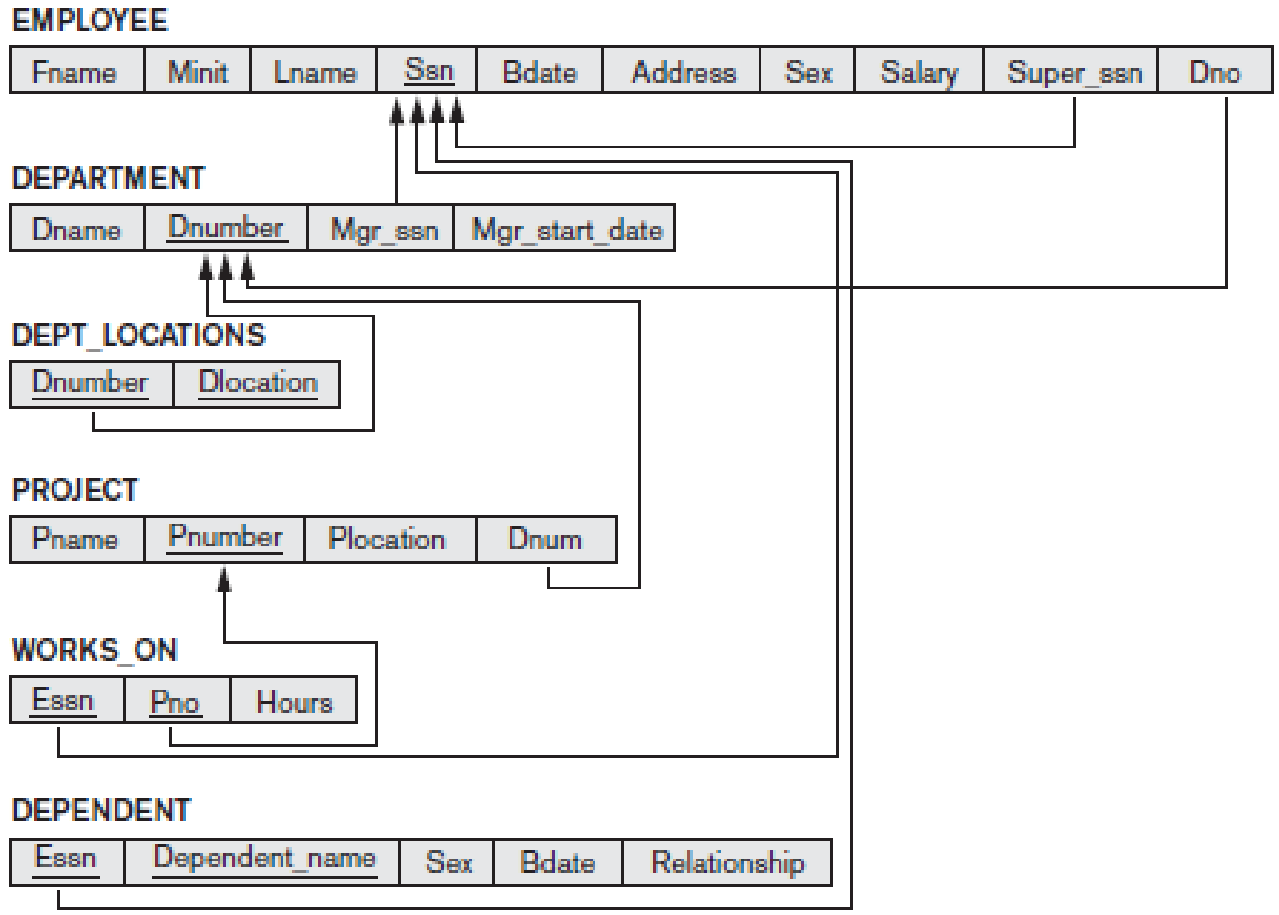
Other Types of Constraints

- Semantic Integrity Constraints:
  - based on application semantics (business rules, standards, constraints) and cannot be expressed by the model
  - Example: “the max. no. of hours per employee for all projects he or she works on is 56 hrs per week”

Displaying a relational database schema and its constraints

- Each relation schema can be displayed as a row of attribute names
- The name of the relation is written above the attribute names
- The primary key attribute (or attributes) will be underlined
- A foreign key (referential integrity) constraints is displayed as a directed arc (arrow) from the foreign key attributes to the referenced table
  - Can also point the primary key of the referenced relation for clarity

COMPANY DB-Relational Model



EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

DEPT_LOCATIONS

Dnumber	Dlocation

PROJECT

Pname	Pnumber	Plocation	Dnum

WORKS_ON

Essn	Pno	Hours

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

Exercise-2-Relational Model

- Consider the following relations for a database that keeps track of business trips of salespersons in a sales office:
  - SALESPERSON(Ssn, Name, Start_year, Dept_no)
  - TRIP(Ssn, From_city, To_city, Departure_date, Return_date, Trip_id)
  - EXPENSE(Trip_id, Account#, Amount)
- A trip can be charged to one or more accounts. Specify the foreign keys for this schema, stating any assumptions you make. Draw the Relational model

Exercise-3-Relational Model

Consider the following relations for a database that keeps track of student enrollment in courses and the books adopted for each course:

STUDENT(SSN, Name, Major, Bdate)

COURSE(Course#, Cname, Dept)

ENROLL(SSN, Course#, Quarter, Grade)

BOOK_ADOPTION(Course#, Quarter, Book_ISBN)

TEXT(Book_ISBN, Book_Title, Publisher, Author)

Draw a relational schema diagram specifying the foreign keys for this schema.

Operations on Relations and constraints

- The following operations update and modify the database
- INSERT a tuple.
- DELETE a tuple.
- MODIFY a tuple.
- Integrity constraints should not be violated by the update operations.
- Several update operations may have to be grouped together.
- Updates may **propagate** (AKA cascading) to cause other updates automatically.
- This may be necessary to maintain integrity constraints.

Copyright @2007 Ramez Elmasri, Shamkant B. Navathe

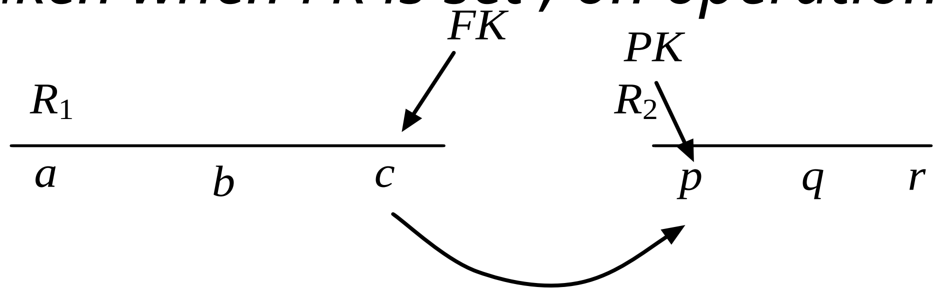
Update Operations on Relations

- In case of integrity violation, several actions can be taken:
  - Cancel the operation that causes the violation (RESTRICT or REJECT option)
  - Perform the operation but inform the user of the violation
  - Trigger additional updates so the violation is corrected (CASCADE option, SET NULL option)
  - Execute a user-specified error-correction routine

Copyright @2007 Ramez Elmasri, Shamkant B. Navathe

Operations on Relations and constraints

Actions need to be taken when FK is set , on operations like update, insert, and delete.



c in R1 refers to *p in R2*

- To insert a tuple in R1 where the value for *c* is not in *p* of R2, then don't allow.

- What if a tuple in R2 is deleted:

  Cascade, don't allow, set to default, set to null.

- What if update on R2's p happens:

  Cascade, don't allow, set to default, set to null.

Operations on Relations and constraints

Operation on relations	**Constraints need to be taken care**
Insert	Null, Not Null, PK, unique, FK, format, Domain
Delete	FK
Update	Null, Not Null, PK, unique, FK, domain, and Semantic

Copyright @2007 Ramez Elmasri, Shamkant B. Navathe

Operations on Relations and constraints

Emp

eid	ename	age	dno
101	Kiran	28	10
105	Gopu	32	20
121	Ahmed	29	20
136	John	28	20
138	Mike	30	30
143	Ali	38	10

Dept

dnum	dname	dloc
10	Payroll	Che
20	Prod	Hyd
30	Acct	Bgl

Op : Care for constraints.

Insert :PK, Unique, null, notnull, format, FK
Update :PK, Unique, null, notnull, format, FK
Delete :No care

Op : Care for constraints.

Insert : PK, Unique, null, notnull, format
Update : PK, Unique, null, notnull, format, FK
Delete : FK

Copyright @2007 Ramez Elmasri, Shamkant B. Navathe

Possible violations for each operation

- INSERT may violate any of the constraints:
  - Domain constraint:
    - if one of the attribute values provided for the new tuple is not of the specified attribute domain
  - Key constraint:
    - if the value of a key attribute in the new tuple already exists in another tuple in the relation
  - Referential integrity:
    - if a foreign key value in the new tuple references a primary key value that does not exist in the referenced relation
  - Entity integrity:
    - if the primary key value is null in the new tuple

Copyright @2007 Ramez Elmasri, Shamkant B. Navathe

Possible violations for each operation

- DELETE may violate only referential integrity:
  - If the primary key value of the tuple being deleted is referenced from other tuples in the database
    - Can be remedied by several actions: RESTRICT, CASCADE, SET NULL
      - RESTRICT option: reject the deletion
      - CASCADE option: propagate the new primary key value into the foreign keys of the referencing tuples
      - SET NULL option: set the foreign keys of the referencing tuples to NULL
  - One of the above options must be specified during database design for each foreign key constraint

Copyright @2007 Ramez Elmasri, Shamkant B. Navathe

Possible violations for each operation

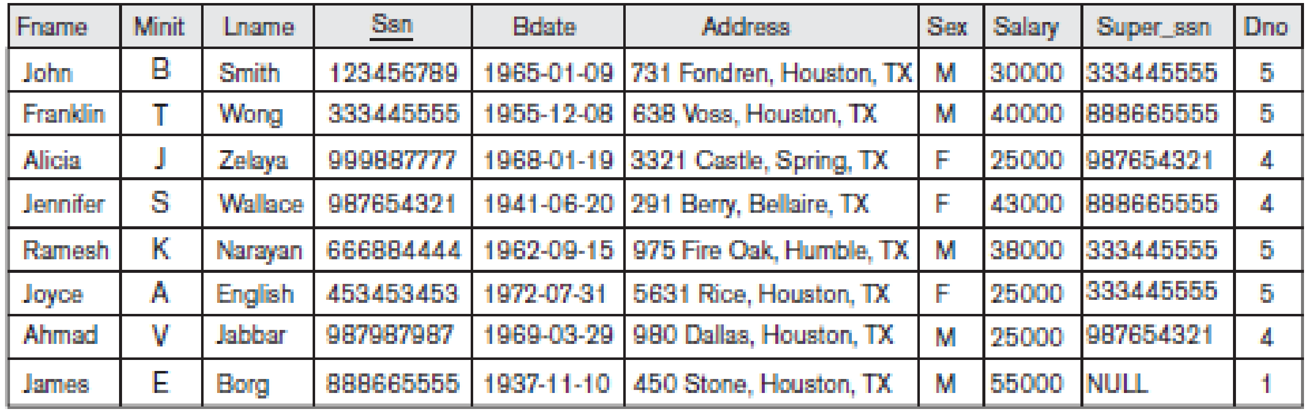
- UPDATE may violate domain constraint and NOT NULL constraint on an attribute being modified
- Any of the other constraints may also be violated, depending on the attribute being updated:
  - Updating the primary key (PK):
    - Similar to a DELETE followed by an INSERT
    - Need to specify similar options to DELETE
  - Updating a foreign key (FK):
    - May violate referential integrity
  - Updating an ordinary attribute (neither PK nor FK):
    - Can only violate domain constraints

Copyright @2007 Ramez Elmasri, Shamkant B. Navathe

Exercise-4

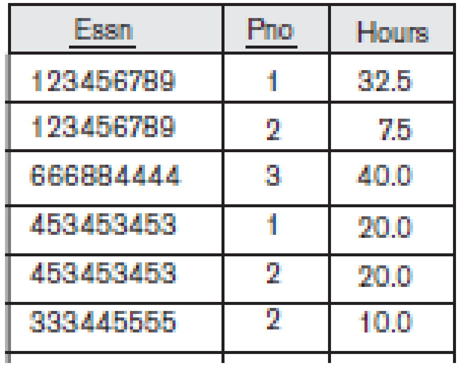
- Suppose that each of the following Update operations is applied directly to the database state shown below. Discuss *all* integrity constraints violated by each operation, if any, and the different ways of enforcing these constraints

EMPLOYEE



Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

WORKS_ON

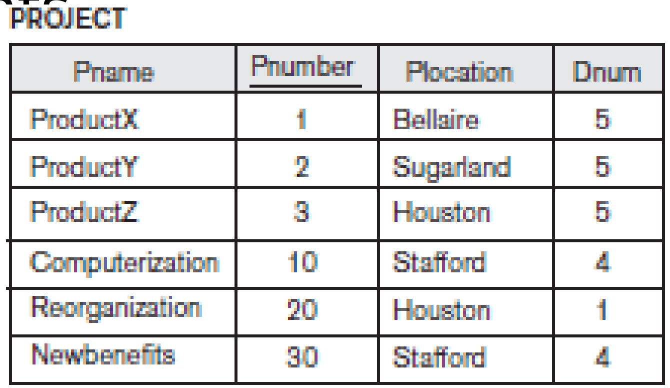


Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0

- Insert <'Robert', 'F', 'Scott', '943775543', '1972-06-21', '2365 Newcastle Rd, Bellaire, TX', M, 58000, '888665555', 1> into EMPLOYEE.
- Delete the EMPLOYEE tuple with Ssn = '123456789'.

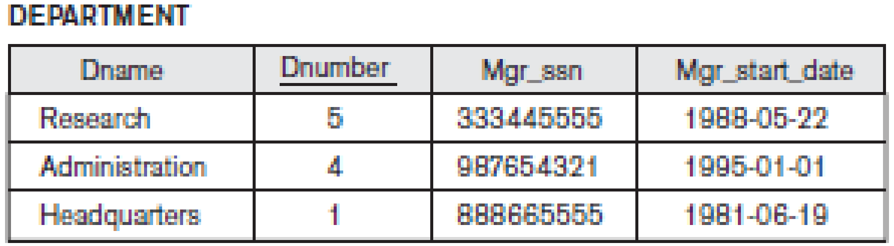
Exercise-5

Suppose that each of the following Update operations is applied directly to the database state shown below. Discuss *all* integrity constraints violated by each operation, if any, and the different ways of enforcing these constraints.



PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4



DEPARTMENT

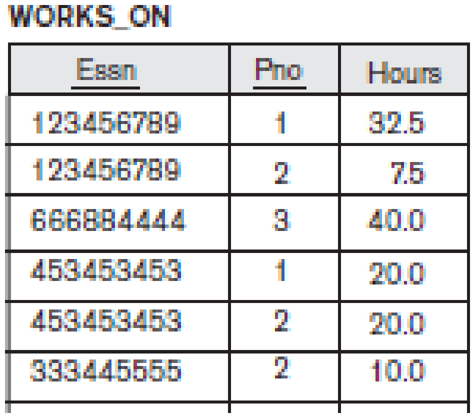
Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

Insert <‘ProductA’, 4, ‘Bellaire’, 2> into PROJECT.

Insert <‘Production’, 4, ‘943775543’, ‘2007-10-01’> into DEPARTMENT.

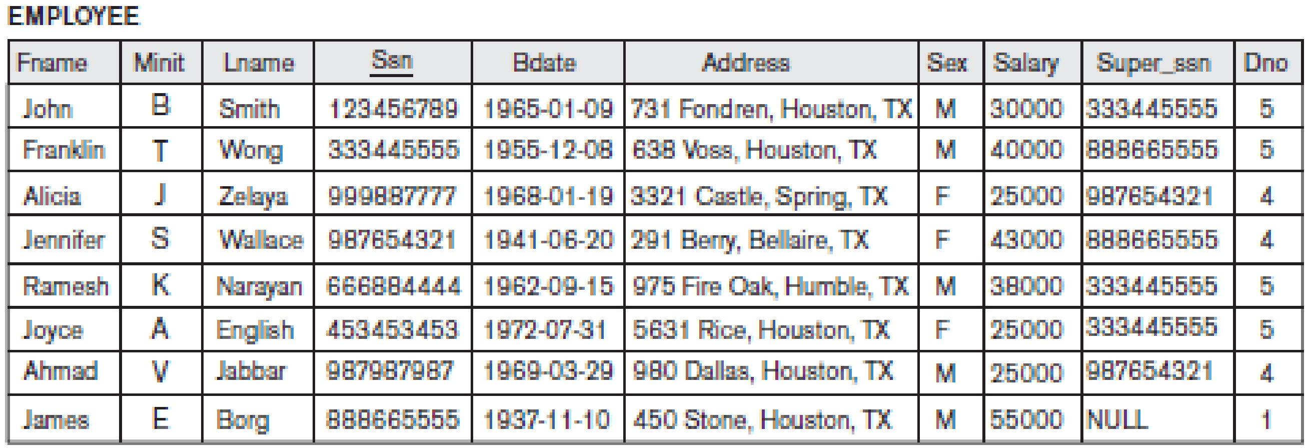
Exercise-6

Suppose that each of the following Update operations is applied directly to the database state shown below. Discuss *all* integrity constraints violated by each operation, if any, and the different ways of enforcing these const



WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0



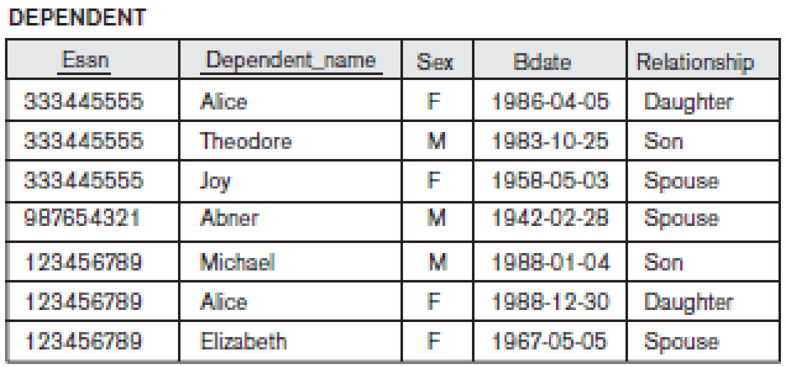
EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

Insert <‘677678989’, NULL, ‘40.0’> into WORKS_ON.

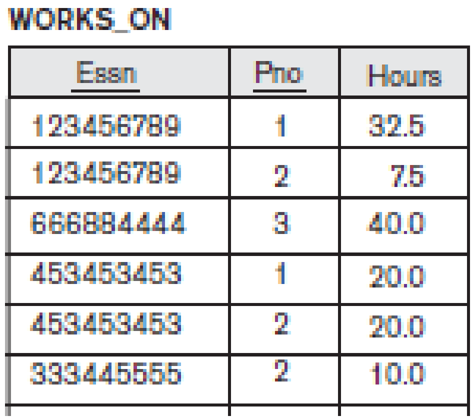
Exercise-7

Suppose that each of the following Update operations is applied directly to the database state shown below. Discuss *all* integrity constraints violated by each operation, if any, and the different ways of enforcing these constraints.



DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse



WORKS_ON

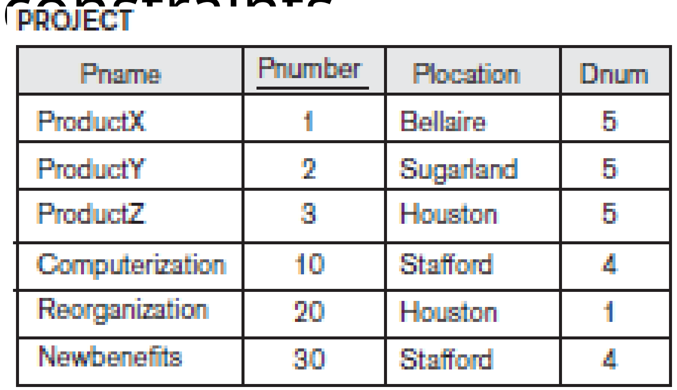
Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0

Insert <‘453453453’, ‘John’, ‘M’, ‘1990-12-12’, ‘spouse’> into DEPENDENT.

Delete the WORKS_ON tuples with Essn = ‘333445555’.

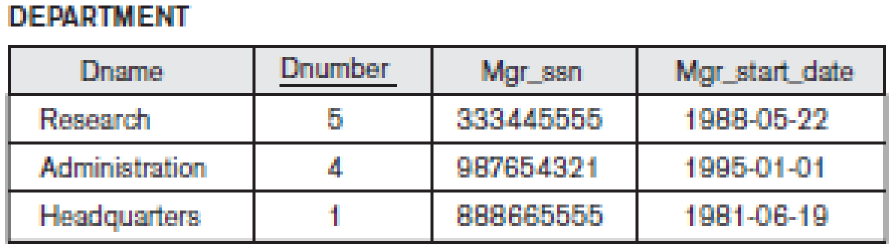
Exercise-8

- Suppose that each of the following Update operations is applied directly to the database state shown below. Discuss *all* integrity constraints violated by each operation, if any, and the different ways of enforcing these constraints



PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4



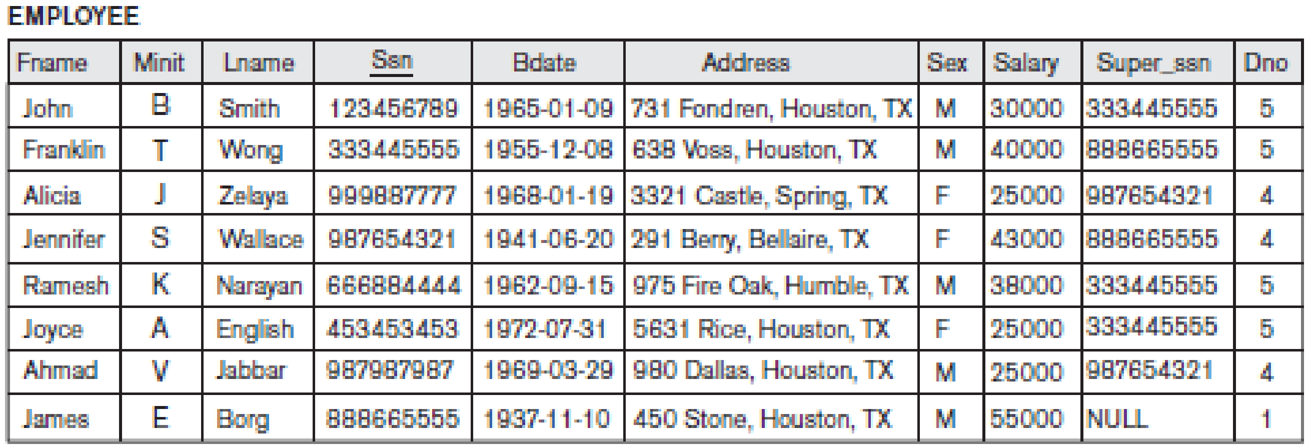
DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

- Delete the PROJECT tuple with Pname = ‘ProductX’.
- Modify the Mgr_ssn and Mgr_start_date of the DEPARTMENT tuple with Dnumber = 5 to ‘123456789’ and ‘2007-10-01’, respectively.

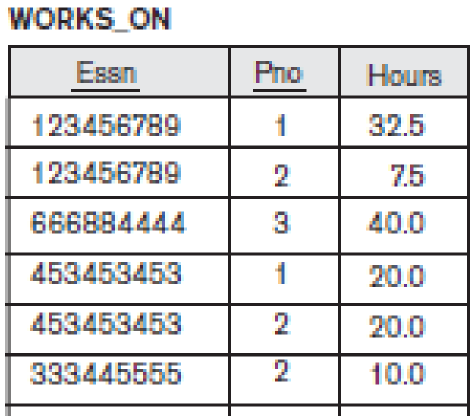
Exercise-9

- Suppose that each of the following Update operations is applied directly to the database state shown below. Discuss *all* integrity constraints violated by each operation, if any, and the different ways of enforcing these constraints



EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

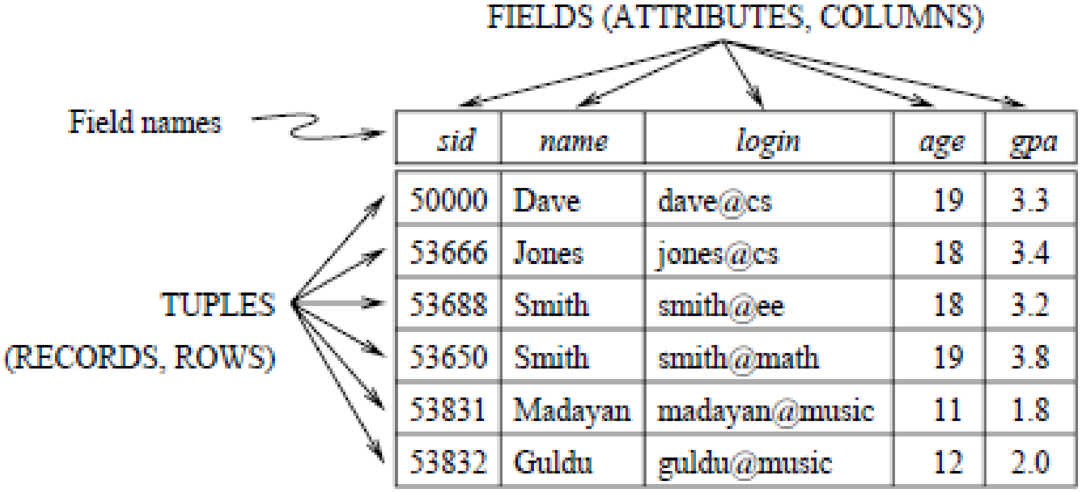


WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0

- Modify the Super_ssn attribute of the EMPLOYEE tuple with Ssn =‘999887777’ to ‘943775543’.
- Modify the Hours attribute of the WORKS_ON tuple with Essn =‘999887777’ and Pno = 10 to ‘5.0’.

Exercise-10


FIELDS (ATTRIBUTES, COLUMNS)
Field names
TUPLES
(RECORDS, ROWS)

sid	*name*	*login*	*age*	*gpa*
50000	Dave	dave@cs	19	3.3
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2
53650	Smith	smith@math	19	3.8
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0

Give an example of an attribute (or set of attributes) that you can deduce is not a candidate key, based on this instance being legal.

Is there any example of an attribute (or set of attributes) that you can deduce is a candidate key, based on this instance being legal?

Summary

- Presented Relational Model Concepts
  - Definitions
  - Characteristics of relations
- Discussed Relational Model Constraints and Relational Database Schemas
  - Domain constraints' - It makes sure that the data value entered for that particular column matches with the data type defined for that column .
  - Key constraints - A unique constraint (also referred to as a unique key constraint) is a rule that forbids duplicate values in one or more columns within a table.
  - A primary key constraint is a column or combination of columns that has the same properties as a unique constraint.
  - Entity integrity - Entity integrity is concerned with ensuring that each row of a table has a unique and non-null primary key value;
  - Referential integrity - You can use a primary key and foreign key constraints to define relationships between tables.
  - The reference from a row in one table to another table must be valid.
- Described the Relational Update Operations and Dealing with Constraint Violations

Database Design

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

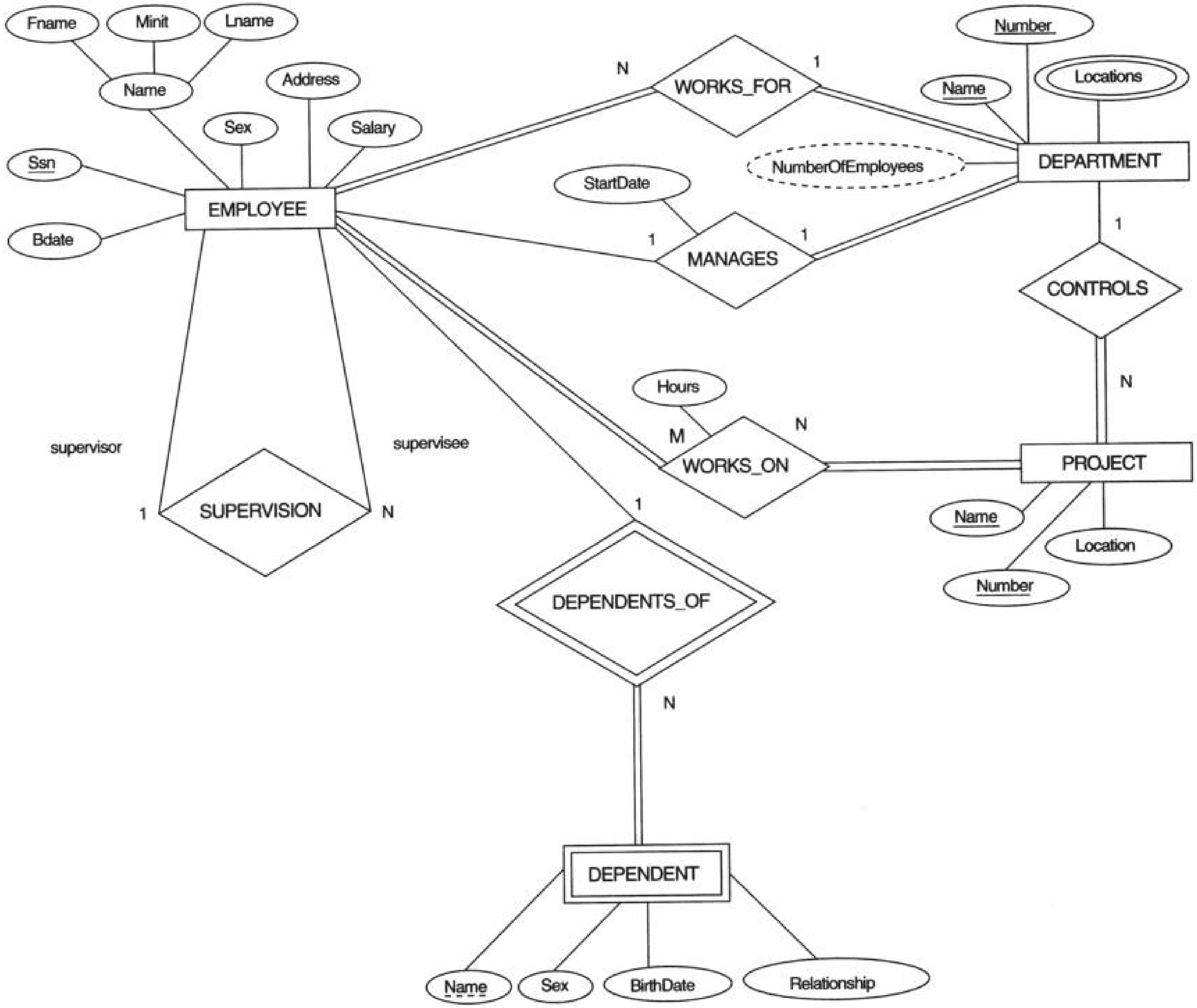
Contact Session 4: ER to Relational Mapping

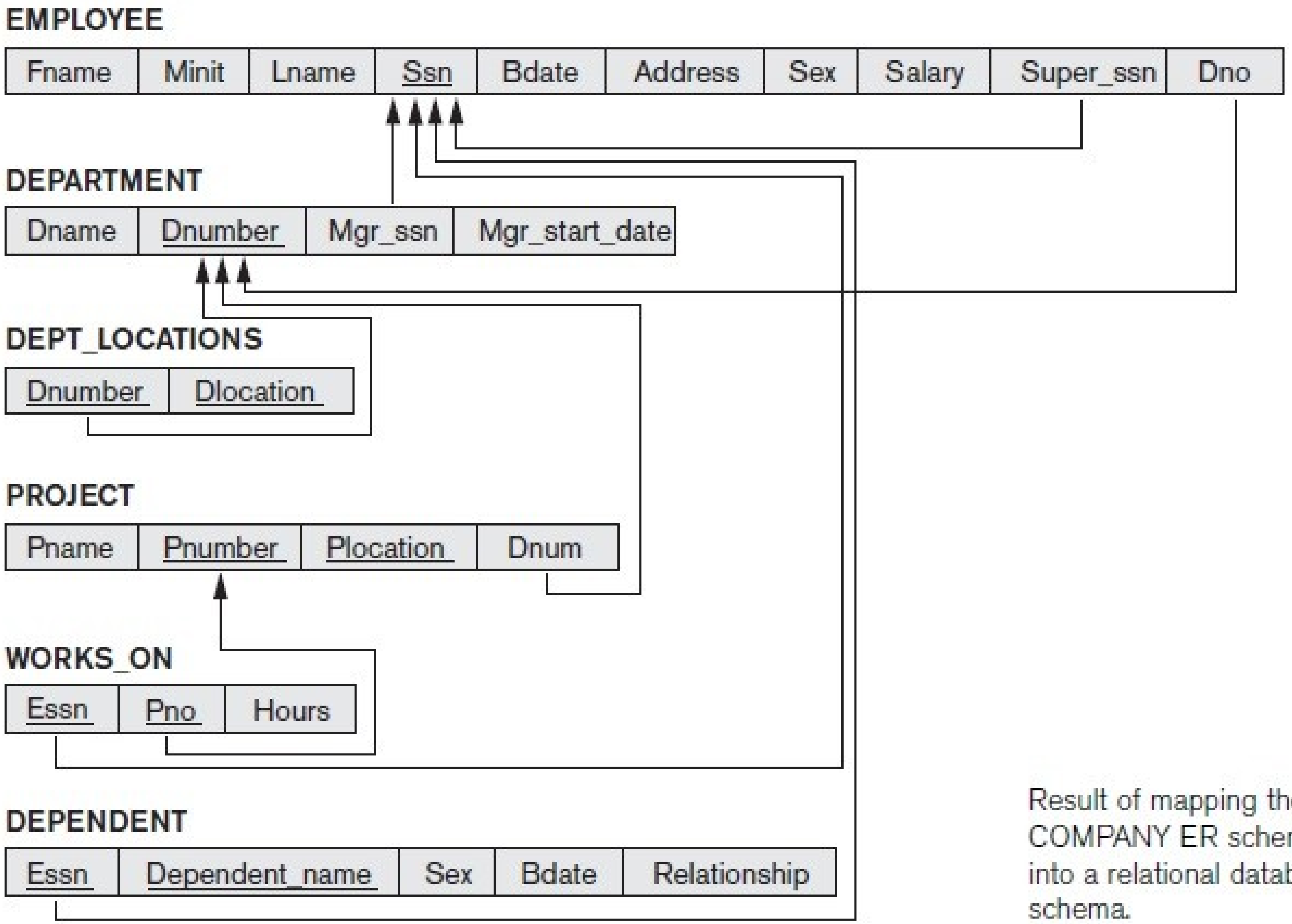
What are we dealing today?

ER to Relational Mapping

1. Mapping rules/guidelines for mapping ER constructs
2. Mapping rules/guidelines for mapping hierarchies
3. Examples

Entity Relationship Model: **COMPANY database**.


Fname
Minit
Lname
Name
Address
Sex
Salary
Ssn
Bdate
EMPLOYEE
N
WORKS_FOR
1
Number
Name
Locations
NumberOfEmployees
DEPARTMENT
StartDate
1
MANAGES
1
1
CONTROLS
N
Hours
M
N
WORKS_ON
PROJECT
Name
Location
Number
supervisor
supervisee
1
SUPERVISION
N
1
DEPENDENTS_OF
N
DEPENDENT
Name
Sex
BirthDate
Relationship



EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

DEPT_LOCATIONS

Dnumber	Dlocation

PROJECT

Pname	Pnumber	Plocation	Dnum

WORKS_ON

Essn	Pno	Hours

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

Result of mapping the COMPANY ER schema into a relational database schema.

ER-to-Relational Mapping Algorithm

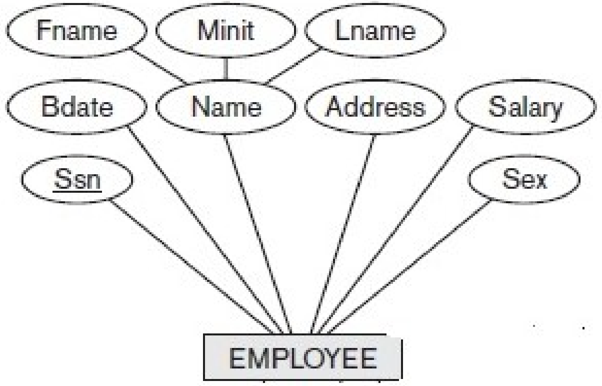
Step 1: Mapping of Regular Entity Types

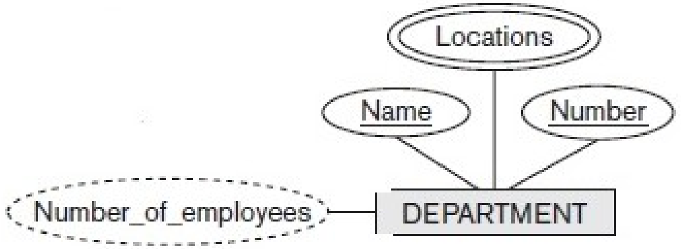
- For each regular (strong) entity type *E* in the ER schema, create a relation *R* that includes all the simple attributes of *E*.
- Include only the simple component attributes of a composite attribute
- Choose one of the key attributes of E as the primary key for R

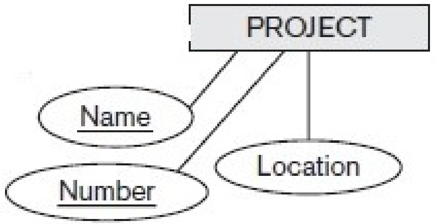
ER-to-Relational Mapping Algorithm

Step 1: Mapping of Regular Entity Types

- If the chosen key of E is a composite, then the set of simple attributes that form it will together form the primary key of R
- If multiple keys were identified for E during the conceptual design, the information describing the attributes that form each additional key is kept in order to specify secondary (unique) keys of relation R.


Fname
Minit
Lname
Bdate
Name
Address
Salary
Ssn
Sex
EMPLOYEE


Locations
Name
Number
Number_of_employees
DEPARTMENT


PROJECT
Name
Number
Location

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary

DEPARTMENT

Dname	Dnumber

PROJECT

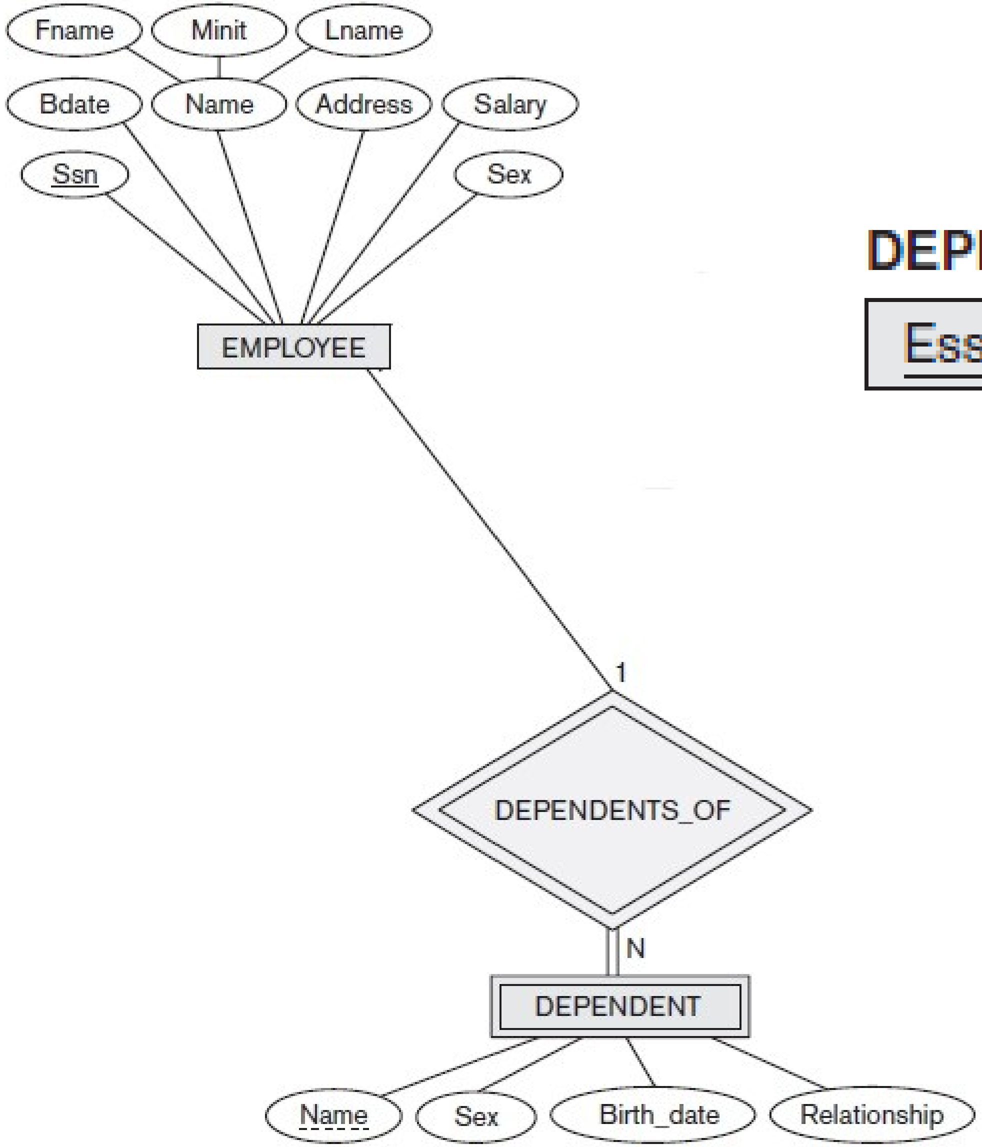
Pname	Pnumber	Plocation

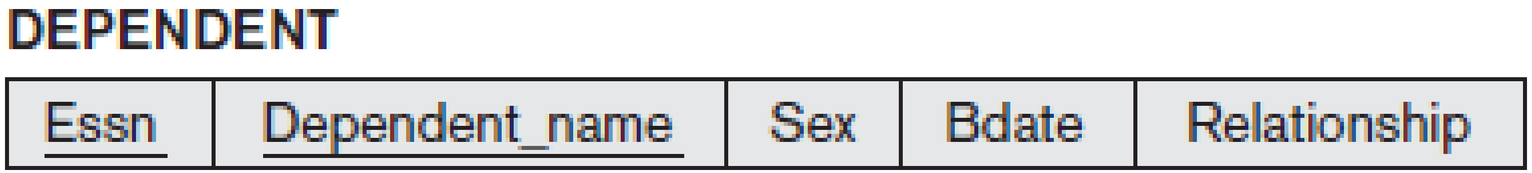
ER-to-Relational Mapping Algorithm

Step 2: Mapping of Weak Entity Types

- For each weak entity type W in the ER schema with owner entity type E, create a relation R and include all simple attributes (or simple components of composite attributes) of W as attributes of R.
- In addition, include primary key of E as foreign key attributes of R (this takes care of mapping the identifying relationship type of W)
- The primary key of R is the combination of the primary key(s) of the owner(s) and the partial key of the weak entity type W(if any)

ER-to-Relational Mapping Algorithm


Fname
Minit
Lname
Bdate
Name
Address
Salary
Ssn
Sex
EMPLOYEE
1
DEPENDENTS_OF
N
DEPENDENT
Name
Sex
Birth_date
Relationship



DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

ER-to-Relational Mapping Algorithm

Step 2: Mapping of Weak Entity Types

- If there is a weak entity type E2 whose owner is also a weak entity type E1, then E1 should be mapped before E2 to determine its primary key first
- It is common to choose the propagate (CASCADE) option for the referential triggered action on the foreign key in the relation corresponding to the weak entity type, since a weak entity has an existence dependency on its owner entity. This can be used for both ON UPDATE and ON DELETE.

ER-to-Relational Mapping Algorithm

Step 3: Mapping of Binary 1:1 Relationship Types

- For each binary 1:1 relationship type R in the ER schema, identify the relations S and T that correspond to the entity types participating in R.
- There are three possible approaches to represent this relationship types:

  1) Foreign key approach

  2) Merged relationship approach

  3) Cross-reference or relationship relation approach

ER-to-Relational Mapping Algorithm

Step 3: Mapping of Binary 1:1 Relationship Types

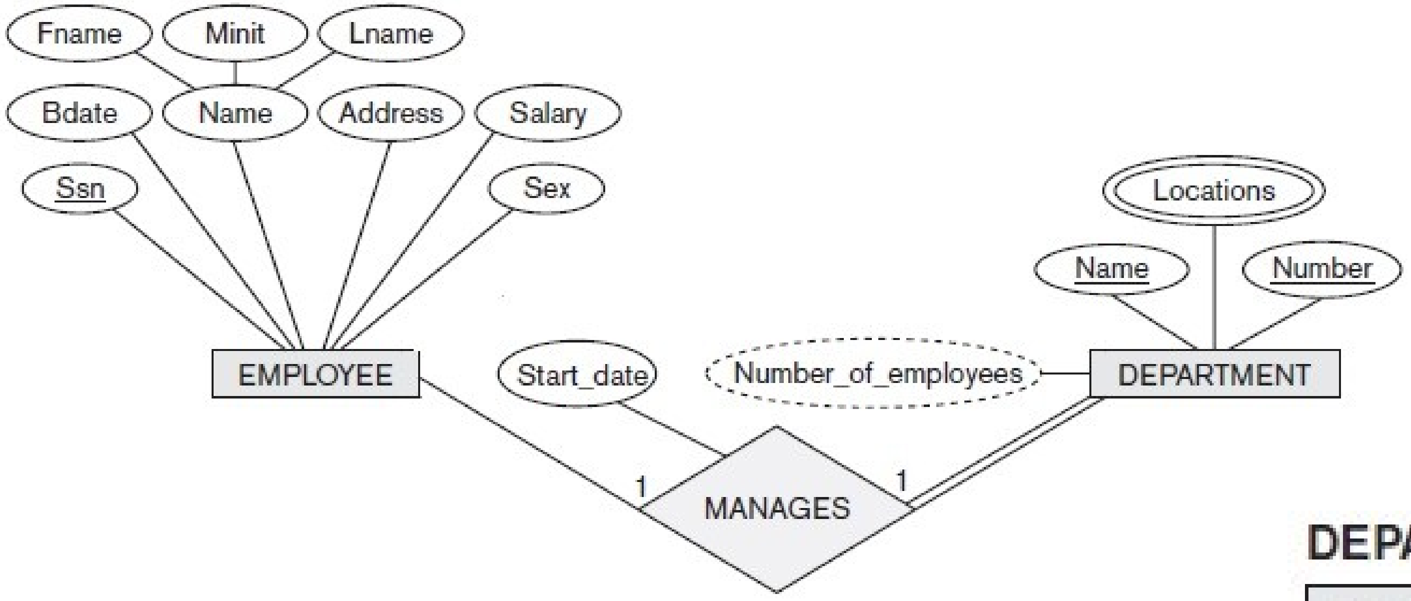
1) Foreign key approach:

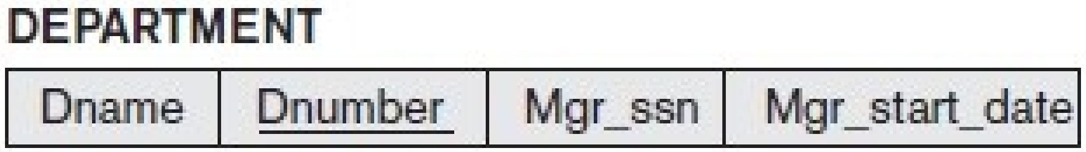
- Choose one of the relations—S, and include as a foreign key in S the primary key of T
- It is better to choose an entity type with total participation in R in the role of S
- Include all the simple attributes of the 1:1 relationship type R as attributes of S.

ER-to-Relational Mapping Algorithm

Step 3: Mapping of Binary 1:1 Relationship Types

1) Foreign key approach:


Fname
Minit
Lname
Bdate
Name
Address
Salary
Ssn
Sex
Locations
Name
Number
EMPLOYEE
Start_date
Number_of_employees
DEPARTMENT
1
MANAGES
1


DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

ER-to-Relational Mapping Algorithm

Step 3: Mapping of Binary 1:1 Relationship Types

2) Merged relationship approach :

- Mapping of a 1:1 relationship type is to merge the two entity types and the relationship into a single relation.
- This is possible when both participations are total, as this would indicate that the two tables will have the exact same number of tuples at all times.

ER-to-Relational Mapping Algorithm

Step 3: Mapping of Binary 1:1 Relationship Types

3) Cross-reference or relationship relation approach :

- set up a third relation R for the purpose of cross-referencing the primary keys of the two relations S and T representing the entity types.
- The relation R is called a relationship relation (or sometimes a lookup table), because each tuple in R represents a relationship instance that relates one tuple from S with one tuple from T.

ER-to-Relational Mapping Algorithm

Step 3: Mapping of Binary 1:1 Relationship Types

3) Cross-reference or relationship relation approach :

- The relation R will include the primary key attributes of S and T as foreign keys.
- The primary key of R will be one of the two foreign keys, and the other foreign key will be a unique key of R.
- The drawback is having an extra relation, and requiring an extra join operation when combining related tuples from the tables.

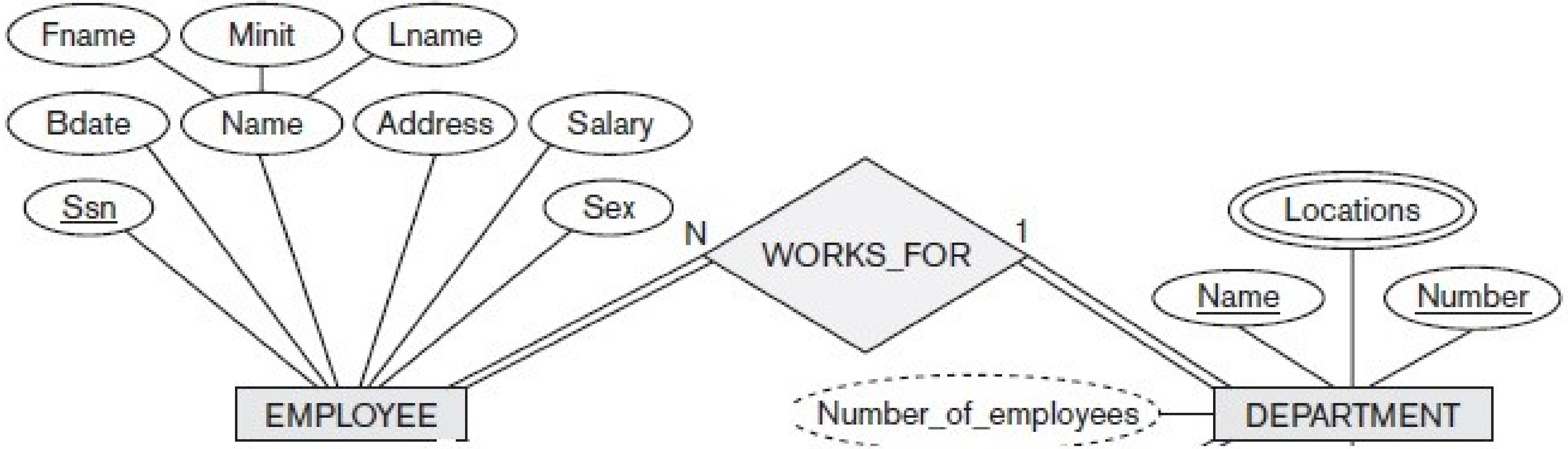
ER-to-Relational Mapping Algorithm

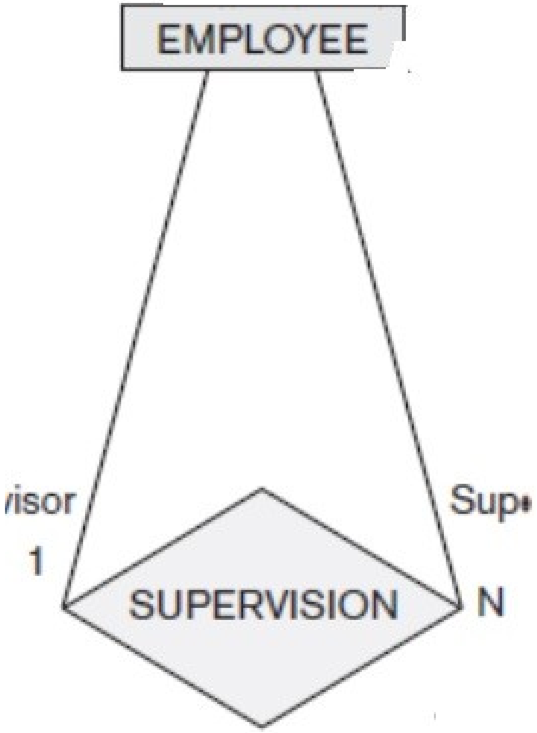
Step 4: Mapping of Binary 1:N Relationship Types

- For each regular binary 1:N relationship type R, identify the relation S that represents the participating entity type at the N-side of the relationship type
- Include as foreign key in S the primary key of the relation T that represents the other entity type participating in R
- Include any simple attributes of the 1:N relationship type as attributes of S.

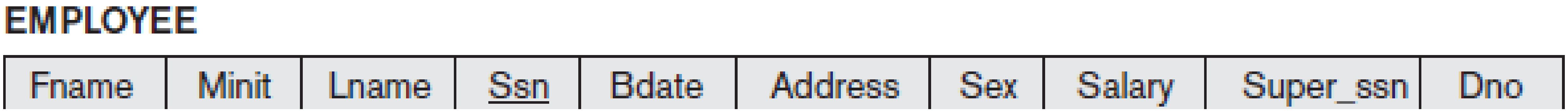
ER-to-Relational Mapping Algorithm

Step 4: Mapping of Binary 1:N Relationship Types


Fname
Minit
Lname
Bdate
Name
Address
Salary
Ssn
Sex
N
WORKS_FOR
1
Locations
Name
Number
EMPLOYEE
Number_of_employees
DEPARTMENT


EMPLOYEE
/isor
1
Sup
SUPERVISION
N

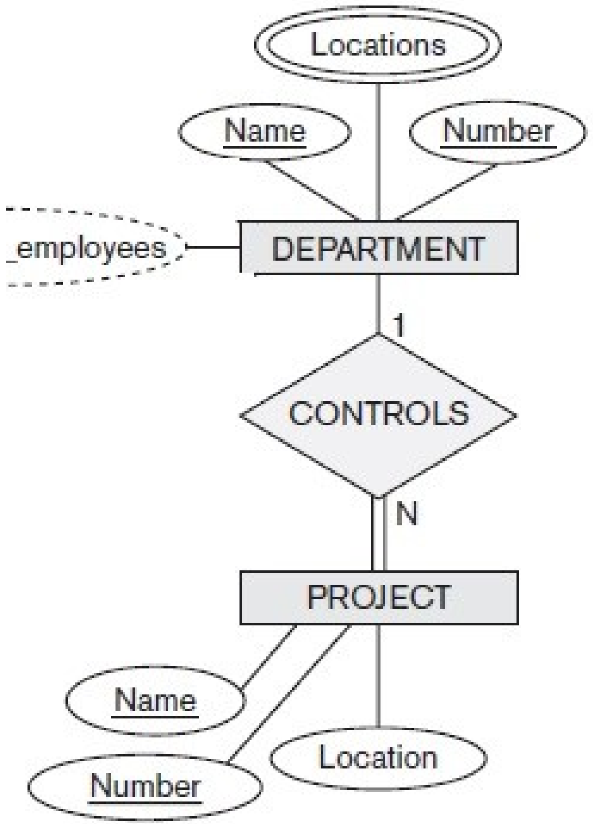
EMPLOYEE



Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

ER-to-Relational Mapping Algorithm

Step 4: Mapping of Binary 1:N Relationship Types


Locations
Name
Number
employees
DEPARTMENT
1
CONTROLS
N
PROJECT
Name
Number
Location

PROJECT

Pname	Pnumber	Plocation	Dnum

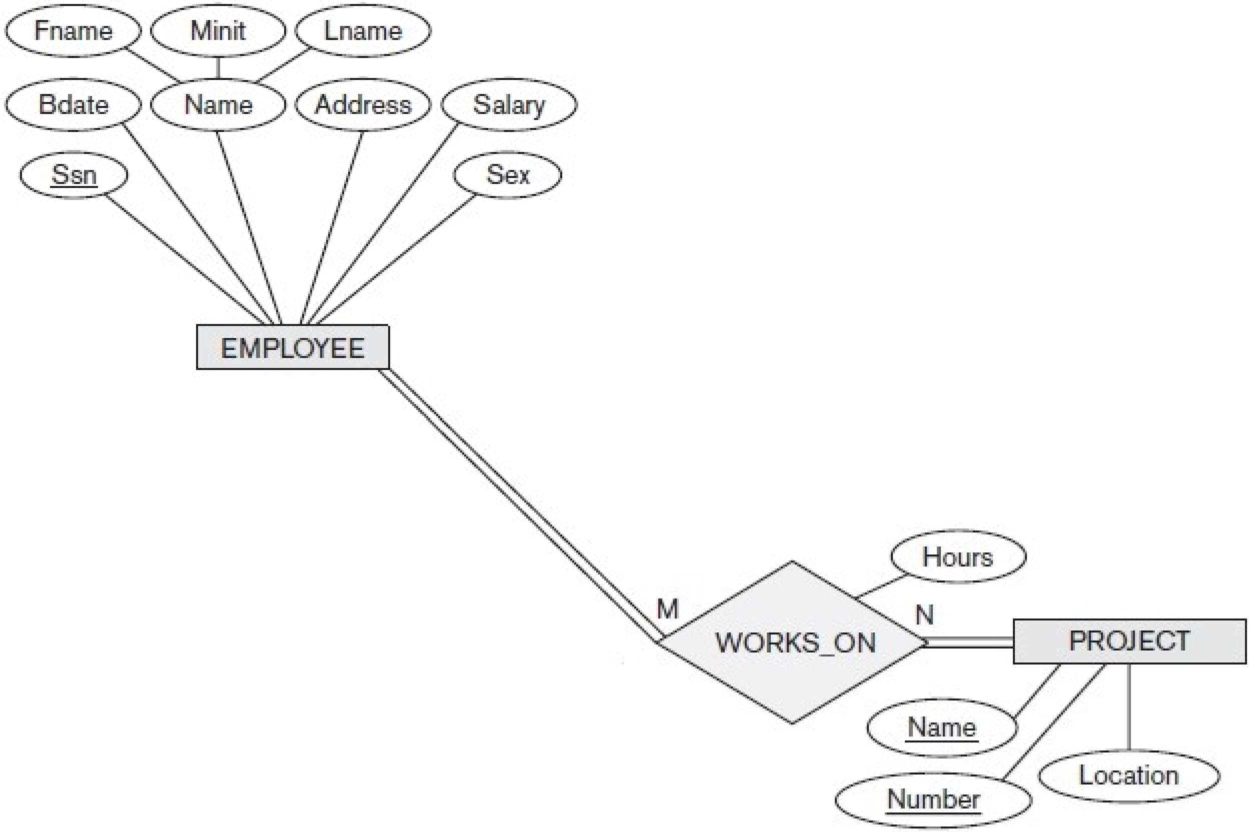
ER-to-Relational Mapping Algorithm

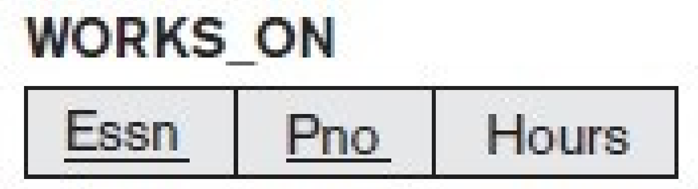
Step 5: Mapping of Binary M:N Relationship Types

- For each binary M:N relationship type R, create a new relation S to represent R.
- Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types
- Their combination will form the primary key of S.
- Also include any simple attributes of the M:N relationship type as attributes of S.
- we must create a separate relationship relation S.

ER-to-Relational Mapping Algorithm

Step 5: Mapping of Binary M:N Relationship


Fname
Minit
Lname
Bdate
Name
Address
Salary
Ssn
Sex
EMPLOYEE
M
WORKS_ON
N
Hours
PROJECT
Name
Number
Location


WORKS_ON

Essn	Pno	Hours

ER-to-Relational Mapping Algorithm

Step 5: Mapping of Binary M:N Relationship Types

- The propagate (CASCADE) option for the referential triggered action should be specified on the foreign keys in the relation corresponding to the relationship R
- This is due to each relationship instance has an existence dependency on each of the entities it relates.
- This can be used for both ON UPDATE and ON DELETE.

ER-to-Relational Mapping Algorithm

Step 5: Mapping of Binary M:N Relationship Types

- Notice that we can always map 1:1 or 1:N relationships in a manner similar to M:N relationships by using the cross-reference (relationship relation) approach.
- This alternative is particularly useful when few relationship instances exist, in order to avoid NULL values in foreign keys.
- In this case, the primary key of the relationship relation will be only one of the foreign keys that reference the participating entity relations.

ER-to-Relational Mapping Algorithm

Step 5: Mapping of Binary M:N Relationship Types

- For a 1:N relationship, the primary key of the relationship relation will be the foreign key that references the entity relation on the N-side.
- For a 1:1 relationship, either foreign key can be used as the primary key of the relationship relation.

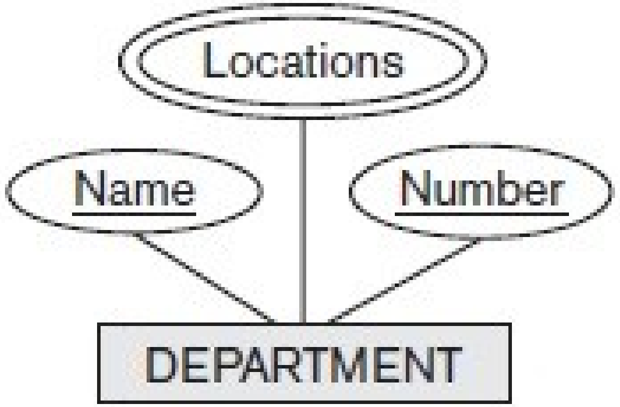
ER-to-Relational Mapping Algorithm

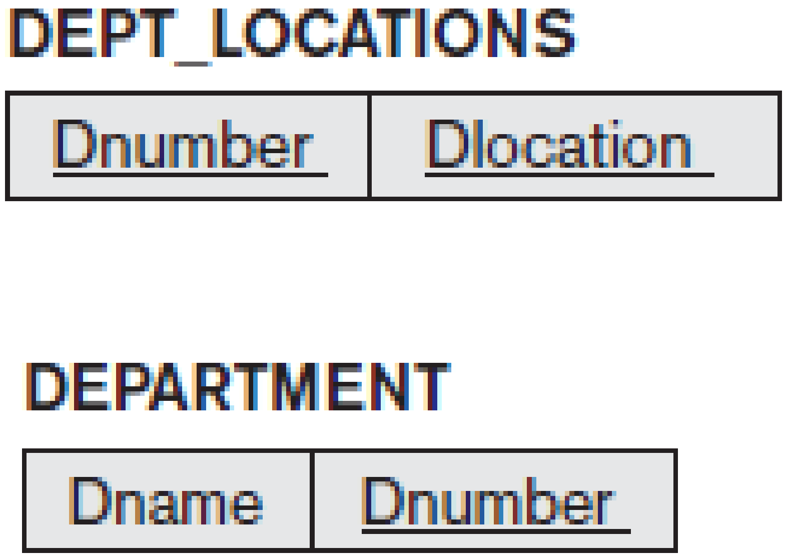
Step 6: Mapping of Multivalued Attributes

- For each multivalued attribute A, create a new relation R.
- This relation R will include an attribute corresponding to A, plus the primary key attribute K—as a foreign key in R—of the relation that represents the entity type or relationship type that has A as a multivalued attribute.
- The primary key of R is the combination of A and K.
- If the multivalued attribute is composite, we include its simple components.

ER-to-Relational Mapping Algorithm

Step 6: Mapping of Multivalued Attributes


Locations
Name
Number
DEPARTMENT


DEPT_LOCATIONS

Dnumber	Dlocation

DEPARTMENT

Dname	Dnumber

ER-to-Relational Mapping Algorithm

Step 6: Mapping of Multivalued Attributes

- The propagate (CASCADE) option for the referential triggered action should be specified on the foreign key in the relation R corresponding to the multivalued attribute for both ON UPDATE and ON DELETE.
- We should also note that the key of R when mapping a composite, multivalued attribute requires some analysis of the meaning of the component attributes.

ER-to-Relational Mapping Algorithm

Step 6: Mapping of Multivalued Attributes

- In some cases, when a multivalued attribute is composite, only some of the component attributes are required to be part of the key of R
- These attributes are similar to a partial key of a weak entity type that corresponds to the multivalued attribute

ER-to-Relational Mapping Algorithm

Step 7: Mapping of N-ary Relationship Types

- For each n-ary relationship type R, where n > 2, create a new relation S to represent R.
- Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types.
- Also include any simple attributes of the n-ary relationship type as attributes of S.

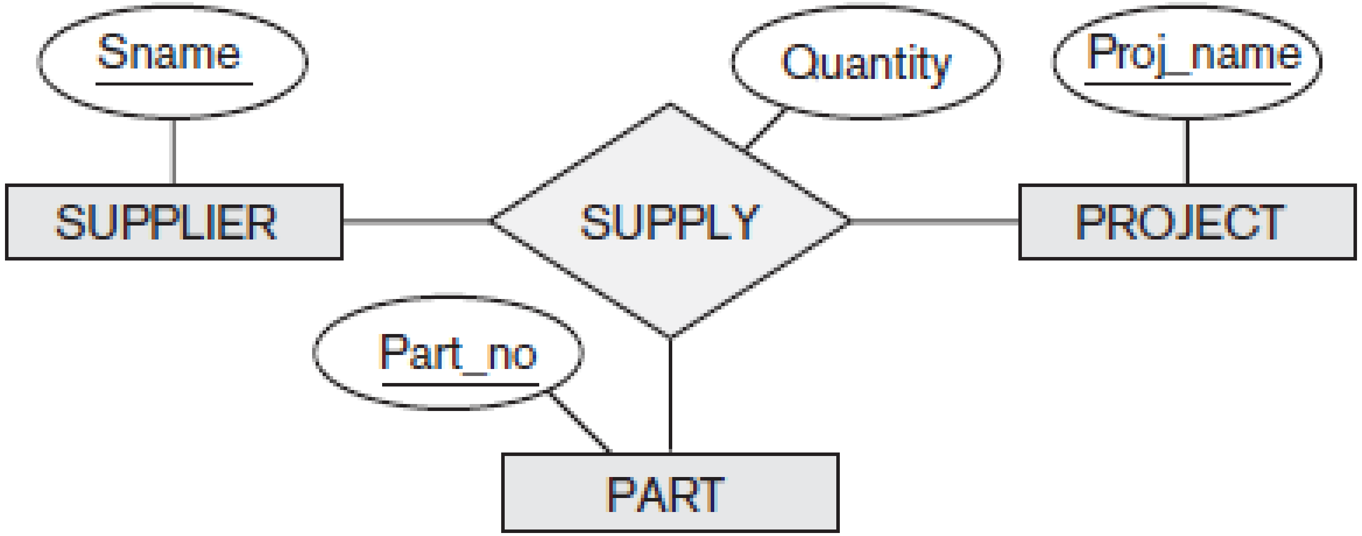
ER-to-Relational Mapping Algorithm

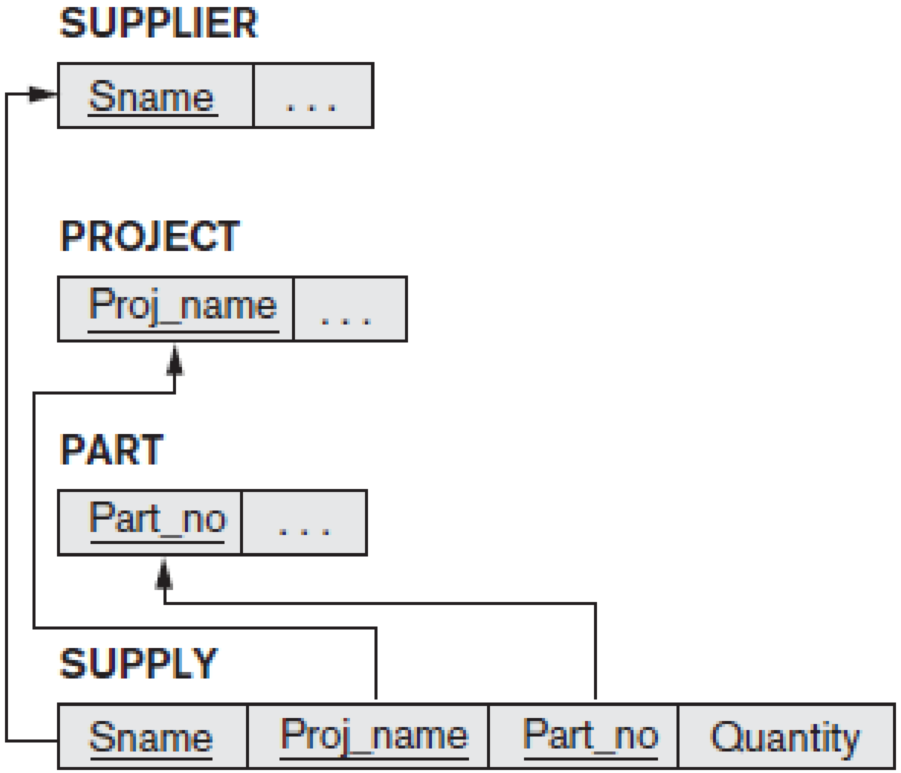
Step 7: Mapping of N-ary Relationship Types

- The primary key of S is usually a combination of all the foreign keys that reference the relations representing the participating entity types.
- However, if the cardinality constraints on any of the entity types E participating in R is 1, then the primary key of S should not include the foreign key attribute that references the relation E corresponding to E

ER-to-Relational Mapping Algorithm

Step 7: Mapping of N-ary Relationship Types


Sname
Quantity
Proj_name
SUPPLIER
SUPPLY
PROJECT
Part_no
PART



SUPPLIER

Sname	. . .

PROJECT

Proj_name	. . .

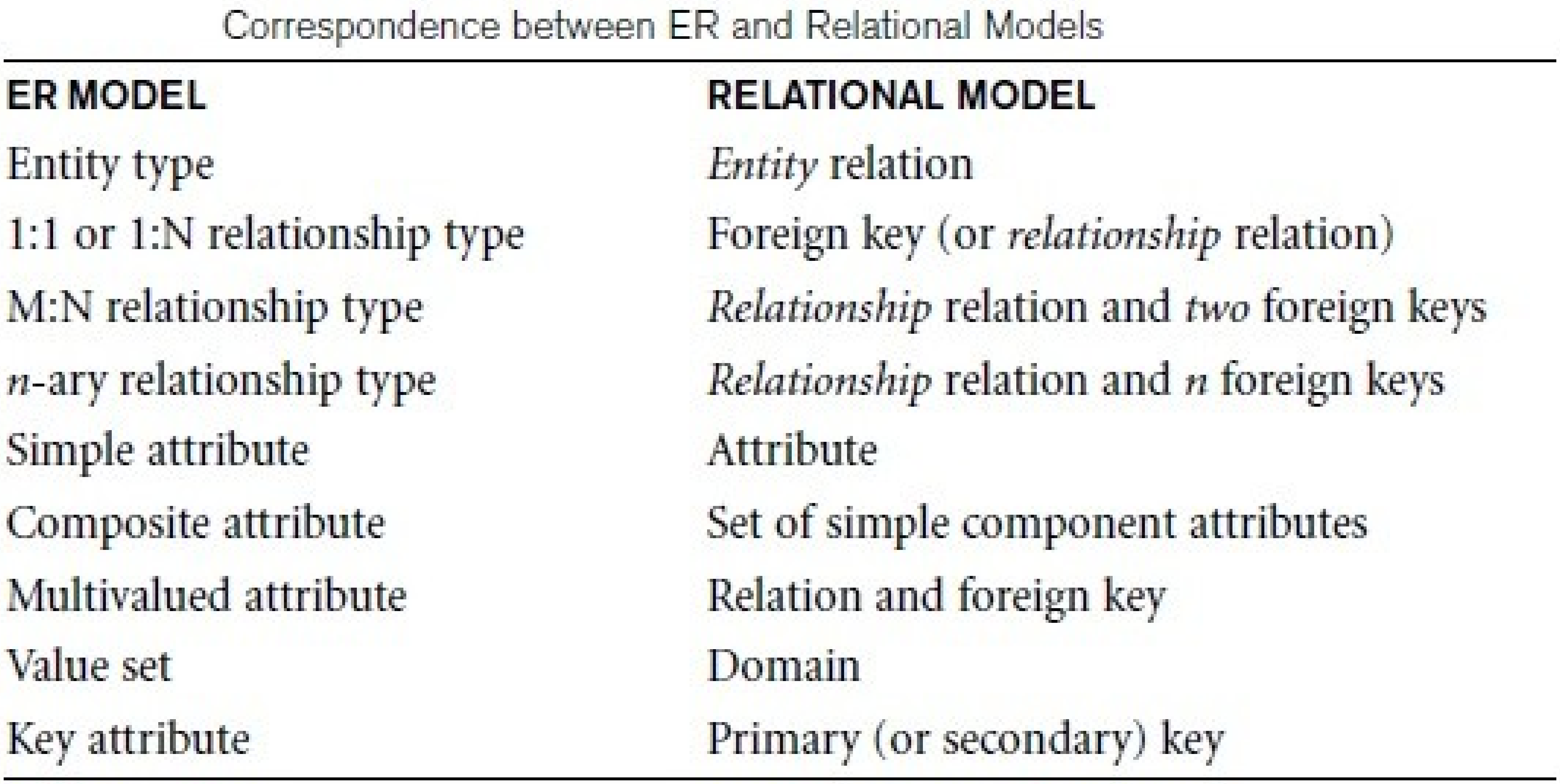
PART

Part_no	. . .

SUPPLY

Sname	Proj_name	Part_no	Quantity

ER-to-Relational Mapping Algorithm: Summary



Correspondence between ER and Relational Models

ER MODEL	RELATIONAL MODEL
Entity type	*Entity* relation
1:1 or 1:N relationship type	Foreign key (or *relationship* relation)
M:N relationship type	*Relationship* relation and *two* foreign keys
n-ary relationship type	*Relationship* relation and *n* foreign keys
Simple attribute	Attribute
Composite attribute	Set of simple component attributes
Multivalued attribute	Relation and foreign key
Value set	Domain
Key attribute	Primary (or secondary) key

Mapping EER Model Constructs to Relations

Step 8: Mapping of Specialization/Generalization

Attrs(*R*) to denote *the attributes of relation R*

PK(*R*) to denote the *primary key of R*

Convert each specialization with m subclasses {S1, S2, ..., Sm} and (generalized) superclass C, where the attributes of C are {k, a1, ...an} and k is the (primary) key, into relation schemas using one of the following options:

8A: Multiple relations—superclass and subclasses.

8B: Multiple relations—subclass relations only.

8C: Single relation with one type attribute.

8D: Single relation with multiple type attributes.

Mapping EER Model Constructs to Relations

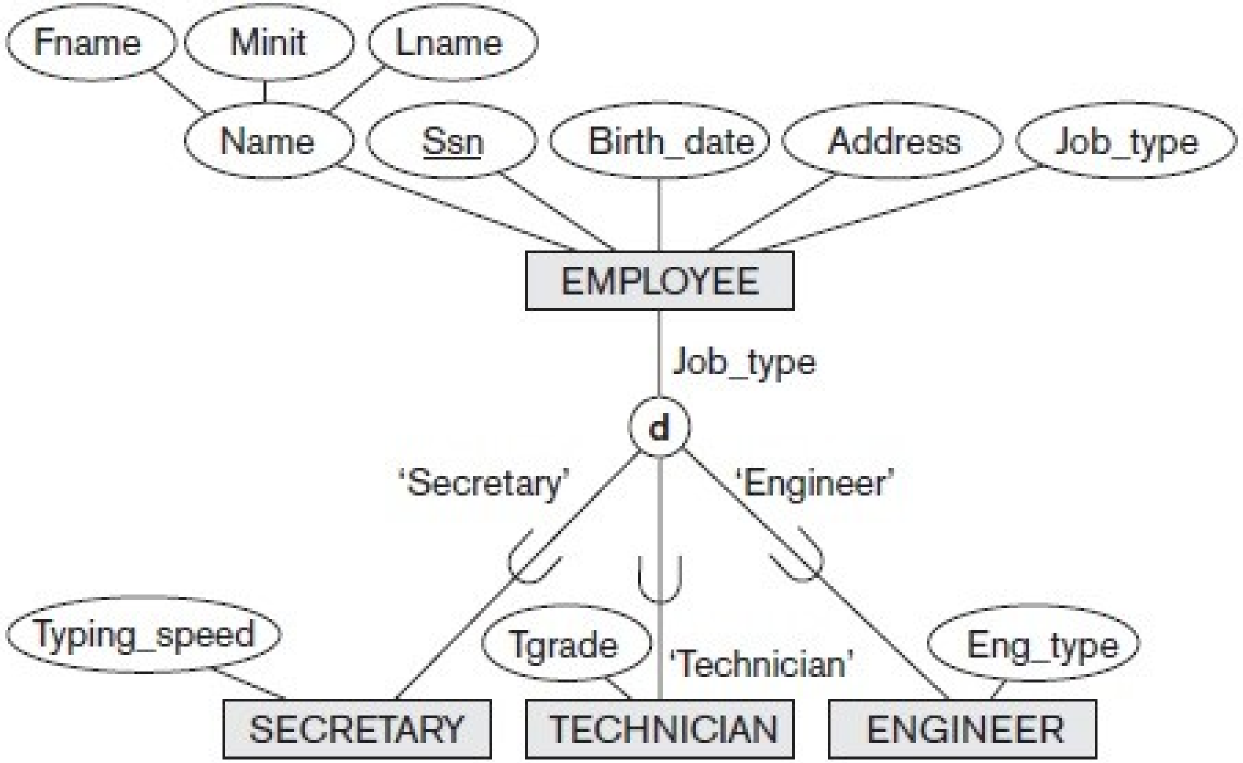
Step 8: Mapping of Specialization/Generalization

8A: Multiple relations—superclass and subclasses.

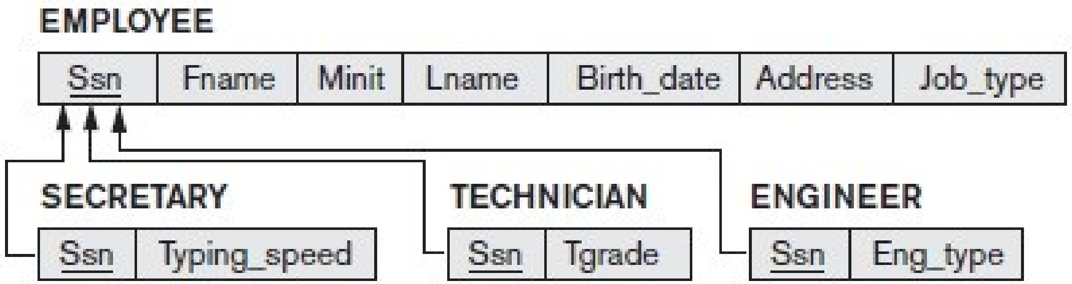
- Create a relation L for C with attributes Attrs(L) = {k, a1, …, an} and PK(L) = k.
- Create a relation Li for each subclass Si, 1≤i≤m , with the attributes Attrs(Li) = {k} ∪ {attributes of Si} and PK(Li) = k.
- This option works for any specialization (total or partial, disjoint or overlapping).

Mapping EER Model Constructs to Relations

Step 8: Mapping of Specialization/Generalization


Fname
Minit
Lname
Name
Ssn
Birth_date
Address
Job_type
EMPLOYEE
Job_type
d
'Secretary'
'Engineer'
'Technician'
Typing_speed
Tgrade
Eng_type
SECRETARY
TECHNICIAN
ENGINEER

ubclasses.



EMPLOYEE

Ssn	Fname	Minit	Lname	Birth_date	Address	Job_type

SECRETARY

Ssn	Typing_speed

TECHNICIAN

Ssn	Tgrade

ENGINEER

Ssn	Eng_type

Mapping EER Model Constructs to Relations

Step 8: Mapping of Specialization/Generalization

8B: Multiple relations— Subclasses relations only.

- Create a relation Li for each subclass Si, 1 ≤ I ≤ m , with the attributes Attrs(Li) = {attributes of Si} ∪ {k, a1, …, an} and PK(Li) = k.
- This option only works for a specialization whose subclasses are total (every entity in the superclass must belong to (at least) one of the subclasses).

Mapping EER Model Constructs to Relations

Step 8: Mapping of Specialization/Generalization

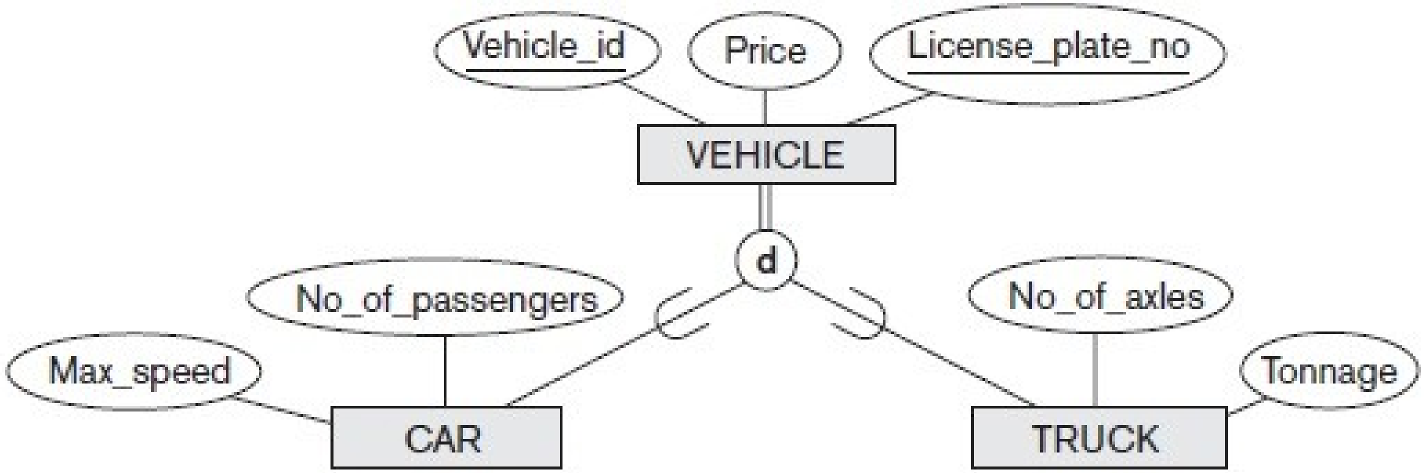
8B: Multiple relations— Subclasses relations only.

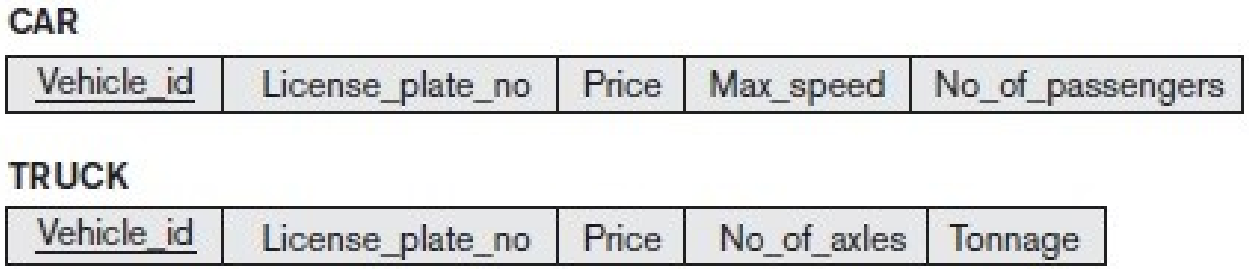
- Additionally, it is only recommended if the specialization has the disjointedness constraint.
- If the specialization is overlapping, the same entity may be duplicated in several relations.

Mapping EER Model Constructs to Relations

Step 8: Mapping of Specialization/Generalization

ly.


Vehicle_id
Price
License_plate_no
VEHICLE
d
No_of_passengers
Max_speed
CAR
No_of_axles
Tonnage
TRUCK



CAR

Vehicle_id	License_plate_no	Price	Max_speed	No_of_passengers

TRUCK

Vehicle_id	License_plate_no	Price	No_of_axles	Tonnage

Mapping EER Model Constructs to Relations

Step 8: Mapping of Specialization/Generalization

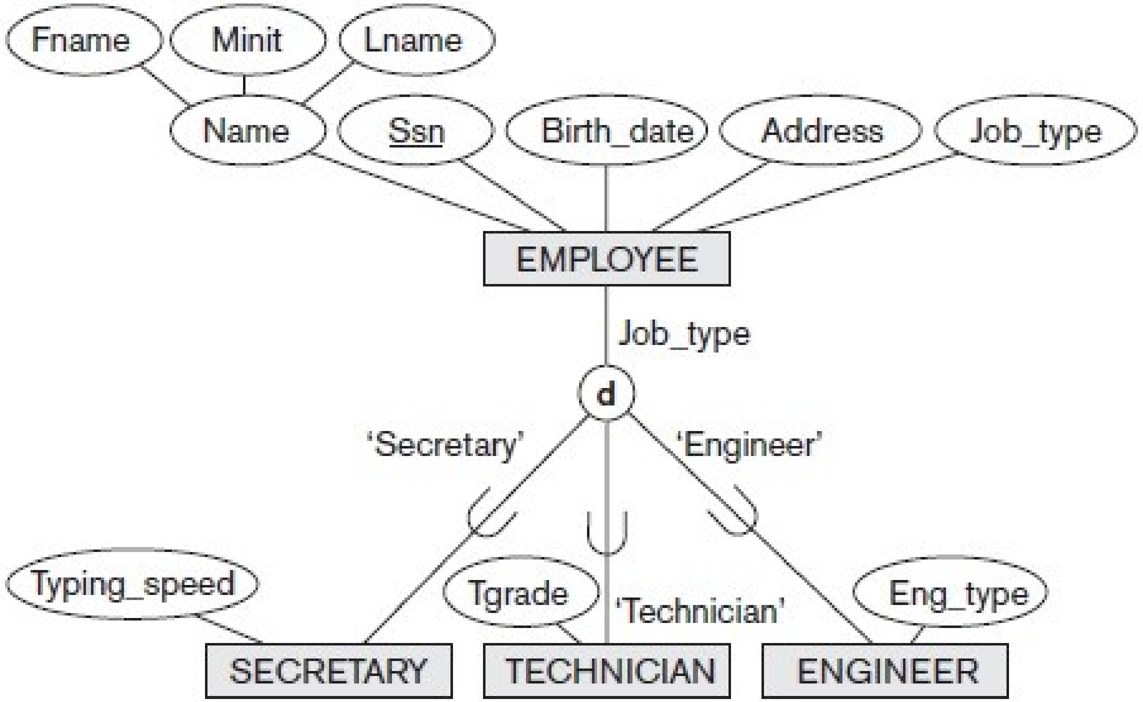
8C: Single relation with one type attribute.

- Create a single relation L with attributes Attrs(L) = {k, a1, …, an} ∪ {attributes of S1} ∪ … ∪ {attributes of Sm} ∪ {t} and PK(L) = k.
- The attribute t is called a type (or discriminating) attribute whose value indicates the subclass to which each tuple belongs, if any.
- This option works only for a specialization whose subclasses are disjoint, and has the potential for

Mapping EER Model Constructs to Relations

Step 8: Mapping of Specialization/Generalization

8C: Single relation with one type


Fname
Minit
Lname
Name
Ssn
Birth_date
Address
Job_type
EMPLOYEE
Job_type
d
'Secretary'
'Engineer'
Typing_speed
Tgrade
'Technician'
Eng_type
SECRETARY
TECHNICIAN
ENGINEER

EMPLOYEE

Ssn	Fname	Minit	Lname	Birth_date	Address	Job_type	Typing_speed	Tgrade	Eng_type

Mapping EER Model Constructs to Relations

Step 8: Mapping of Specialization/Generalization

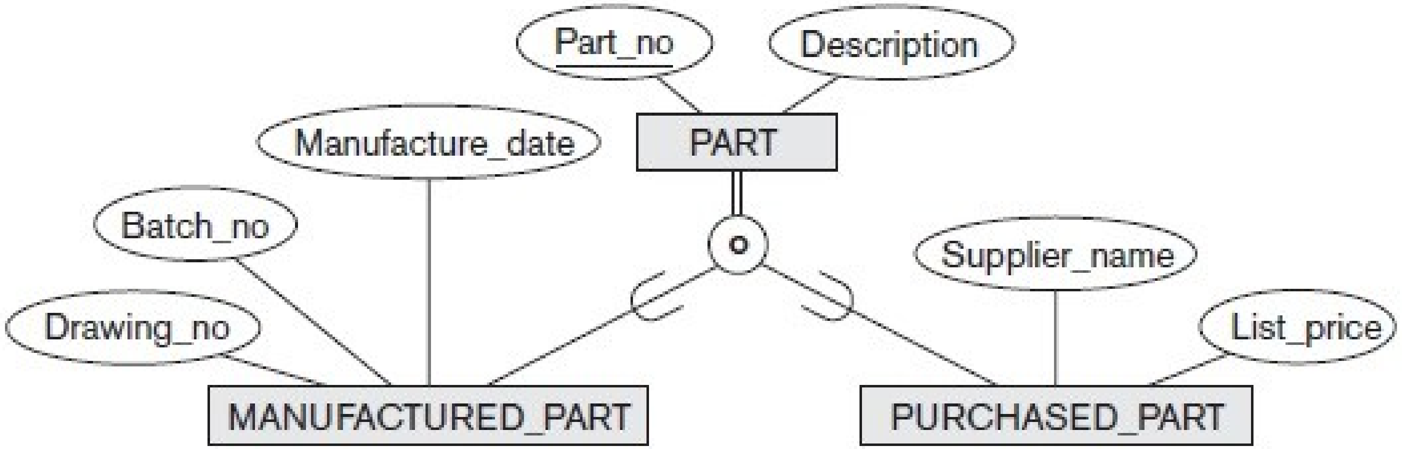
8D: Single relation with multiple type attributes.

- Create a single relation schema L with attributes Attrs(L) = {k, a1, …, an} ∪ {attributes of S1} ∪ … ∪ {attributes of Sm} ∪ {t1, t2, …, tm} and PK(L) = k.
- Each ti, 1 ≤ i ≤ m, is a Boolean type attribute indicating whether a tuple belongs to subclass Si.
- This option is used for a specialization whose subclasses are overlapping (but will also work for a disjoint specialization).

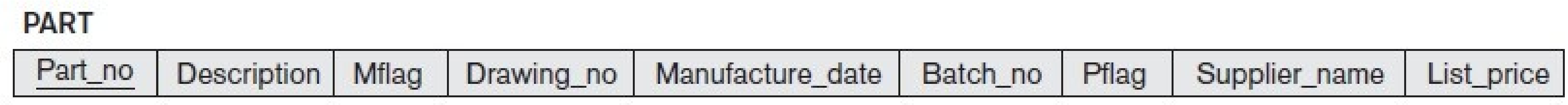
Mapping EER Model Constructs to Relations

Step 8: Mapping of Specialization/Generalization

8D: Single relation with multiple type attributes



Part_no
Description
PART
Manufacture_date
Batch_no
o
Supplier_name
Drawing_no
List_price
MANUFACTURED_PART
PURCHASED_PART


PART

Part_no	Description	Mflag	Drawing_no	Manufacture_date	Batch_no	Pflag	Supplier_name	List_price

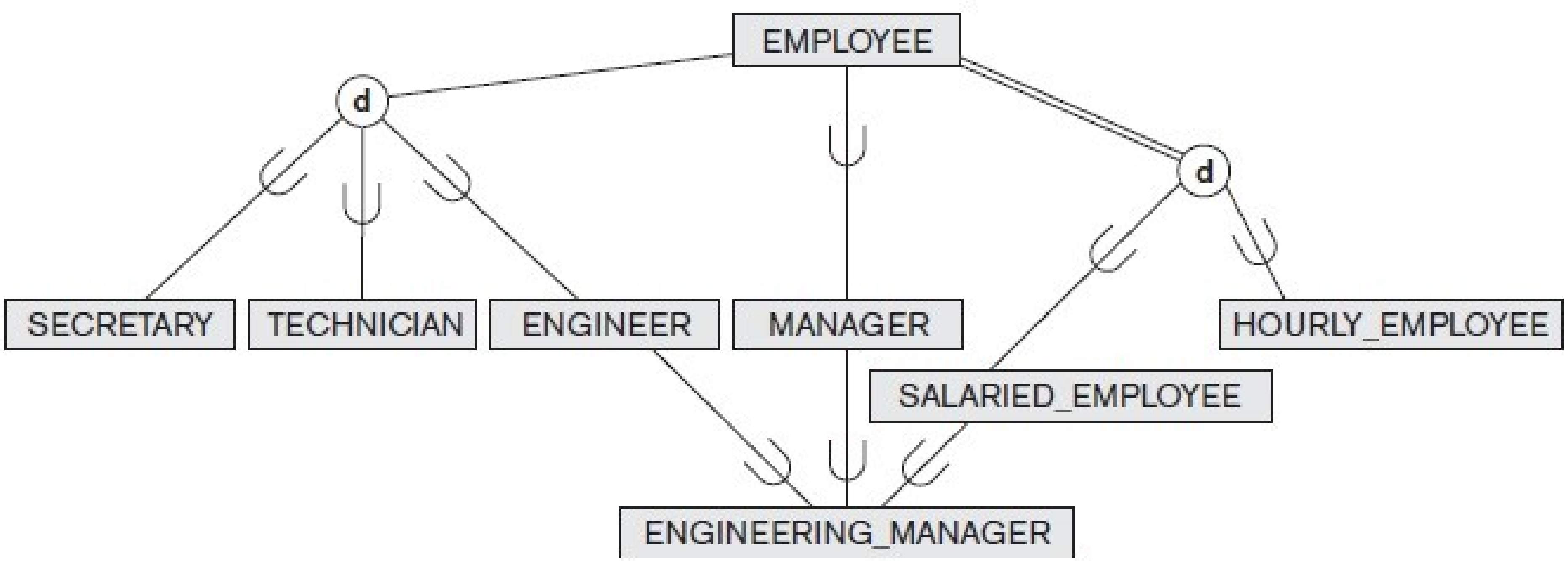
Mapping EER Model Constructs to Relations

Mapping of Shared Subclasses (Multiple Inheritance)

- A shared subclass, such as ENGINEERING_MANAGER in Figure 8.6, is a subclass of several superclasses, indicating multiple inheritance.
- These classes must all have the same key attribute; otherwise, the shared subclass would be modeled as a category (union type)
- We can apply any of the options discussed in step 8 to a shared subclass, subject to the restrictions discussed in step 8 of the mapping algorithm

Mapping EER Model Constructs to Relations

Mapping of Shared Subclasses (Multiple


EMPLOYEE
d
d
SECRETARY
TECHNICIAN
ENGINEER
MANAGER
HOURLY_EMPLOYEE
SALARIED_EMPLOYEE
ENGINEERING_MANAGER

Mapping EER Model Constructs to Relations

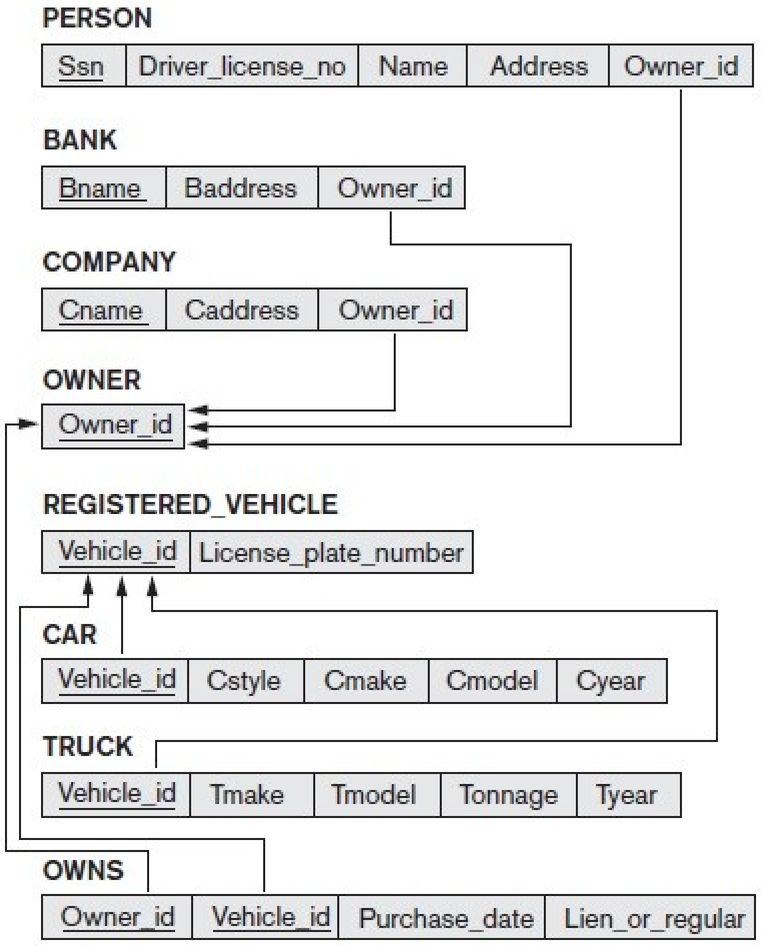
Step 9: Mapping of Categories (Union Types)

- A category (or union type) is a subclass of the union of two or more superclasses that can have different keys because they can be of different entity types
- For mapping a category whose defining superclasses have different keys, it is customary to specify a new key attribute, called a surrogate key, when creating a relation to correspond to the category.

Mapping EER Model Constructs

Step 9: Mapping of Categori

(Union Type



PERSON

Ssn	Driver_license_no	Name	Address	Owner_id

BANK

Bname	Baddress	Owner_id

COMPANY

Cname	Caddress	Owner_id

OWNER

Owner_id

REGISTERED_VEHICLE

Vehicle_id	License_plate_number

CAR

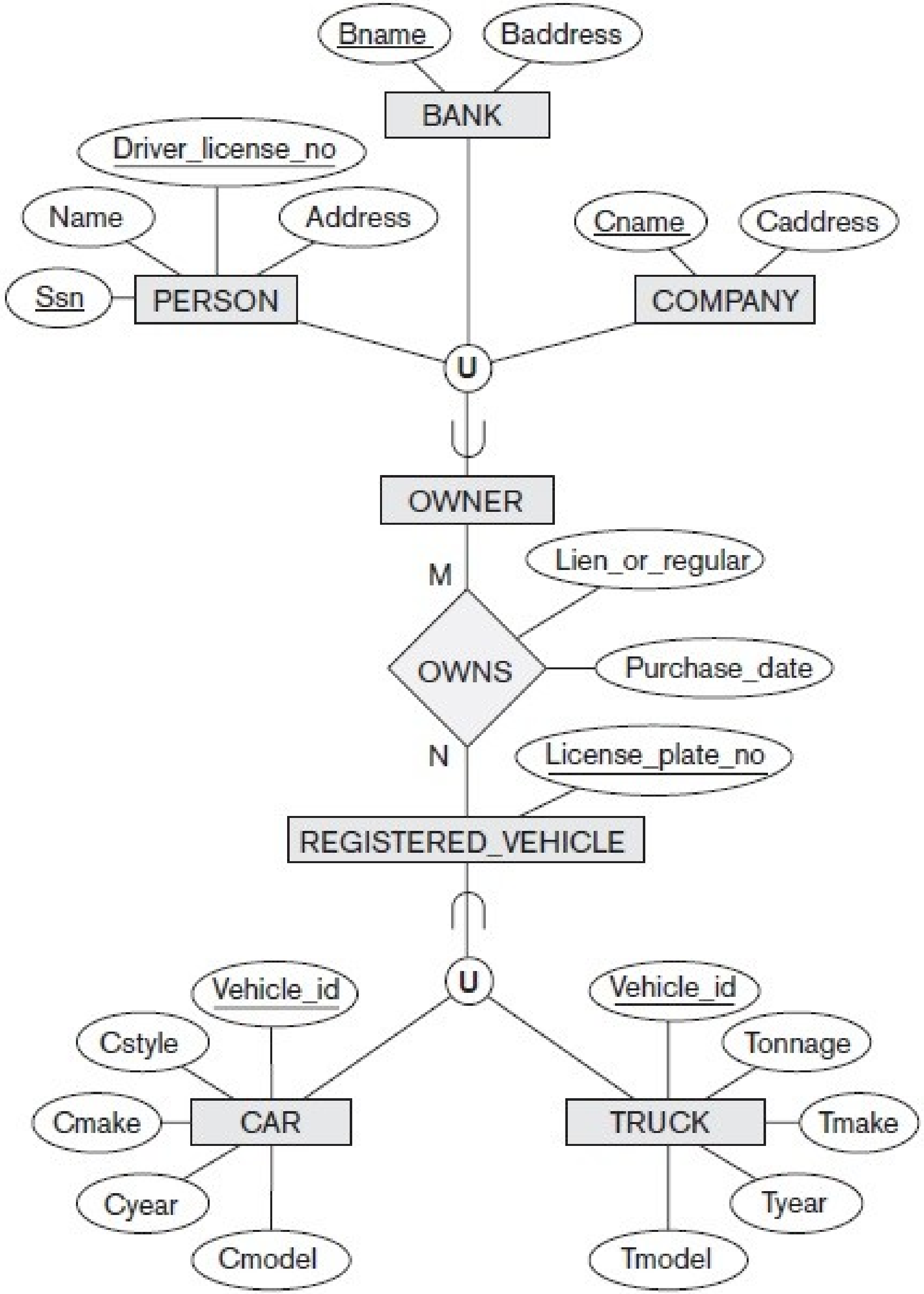
Vehicle_id	Cstyle	Cmake	Cmodel	Cyear

TRUCK

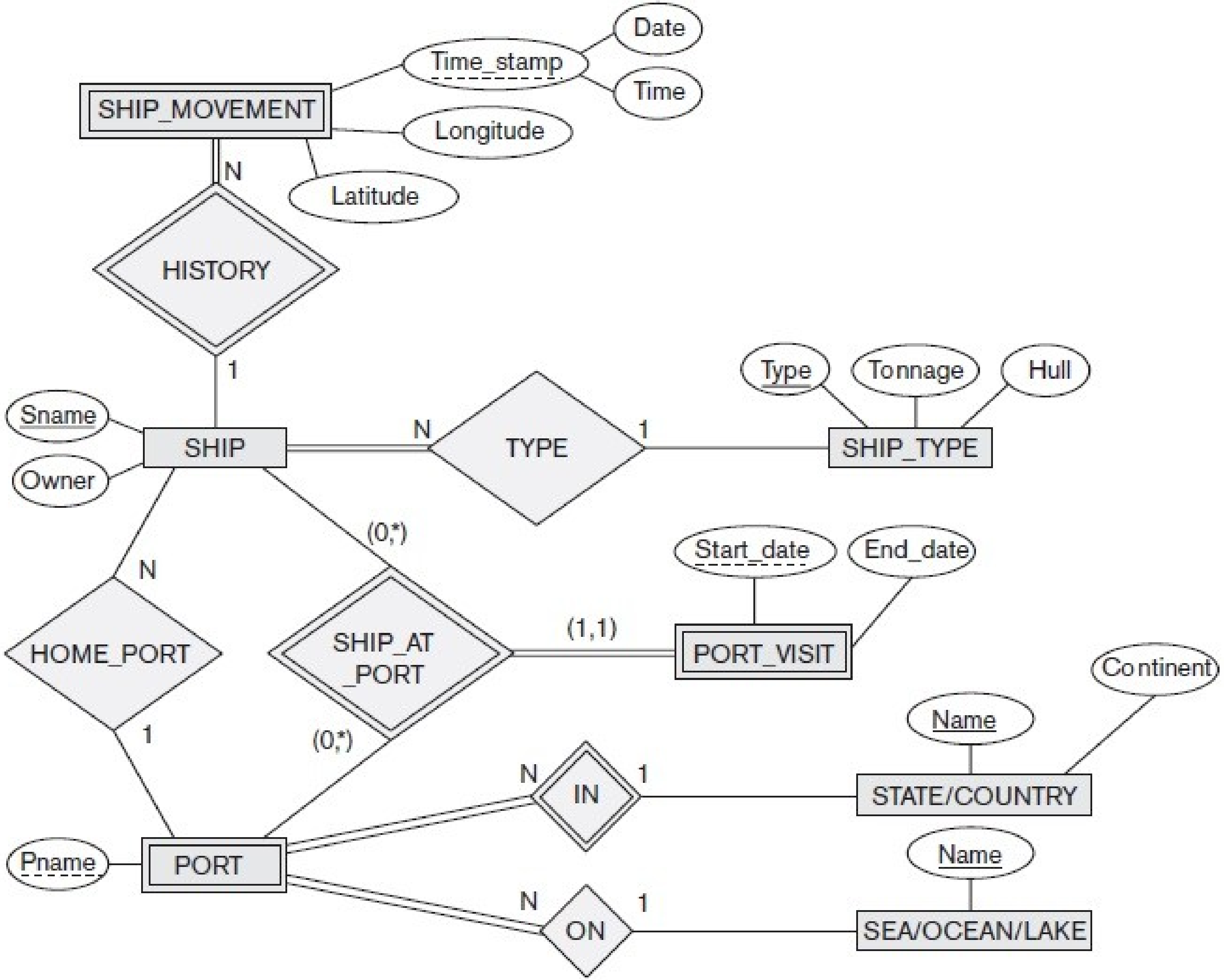
Vehicle_id	Tmake	Tmodel	Tonnage	Tyear

OWNS

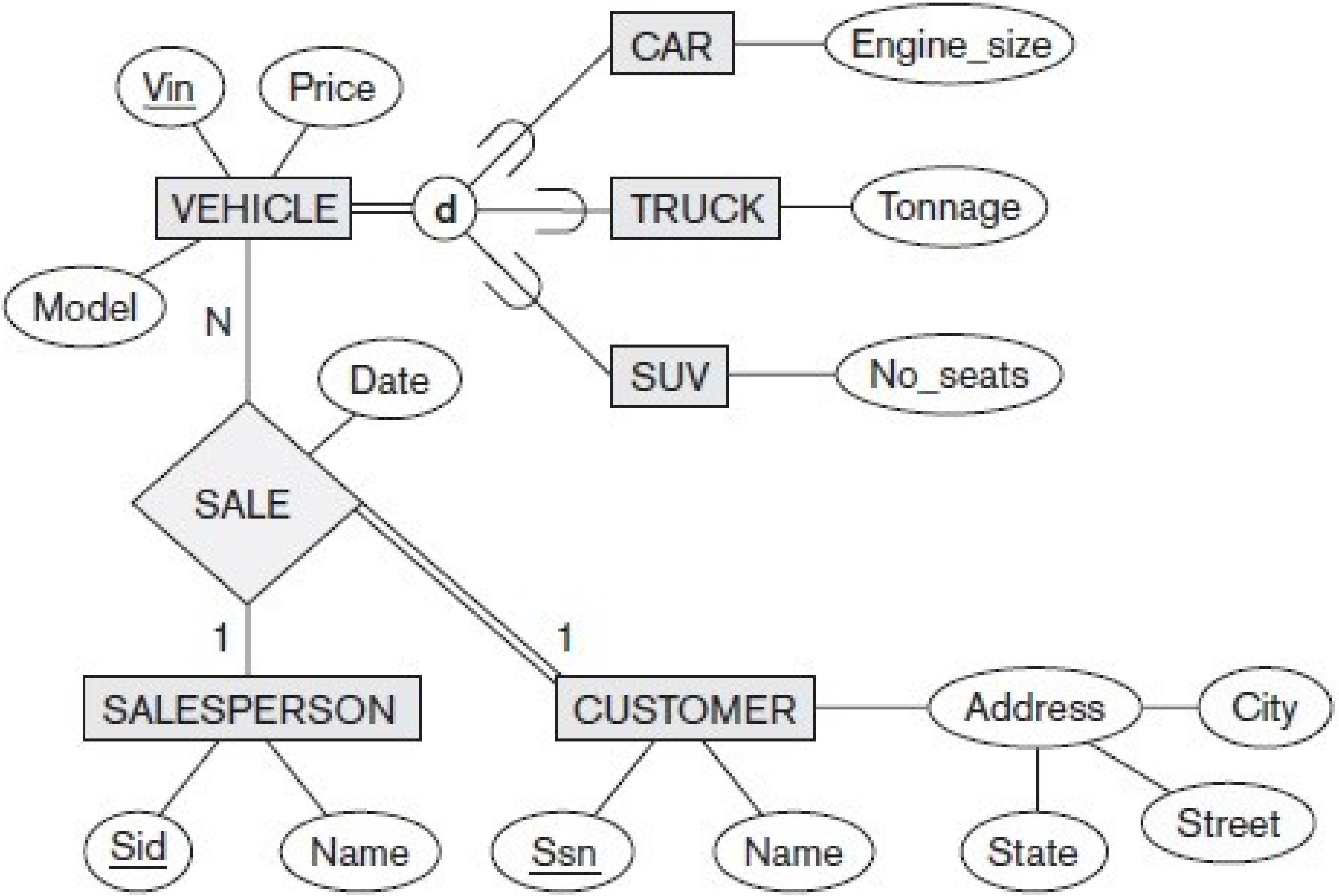
Owner_id	Vehicle_id	Purchase_date	Lien_or_regular


Bname
Baddress
BANK
Driver_license_no
Name
Address
Cname
Caddress
Ssn
PERSON
COMPANY
U
OWNER
M
Lien_or_regular
OWNS
Purchase_date
N
License_plate_no
REGISTERED_VEHICLE
U
Vehicle_id
Vehicle_id
Cstyle
Tonnage
Cmake
CAR
TRUCK
Tmake
Cyear
Tyear
Cmodel
Tmodel

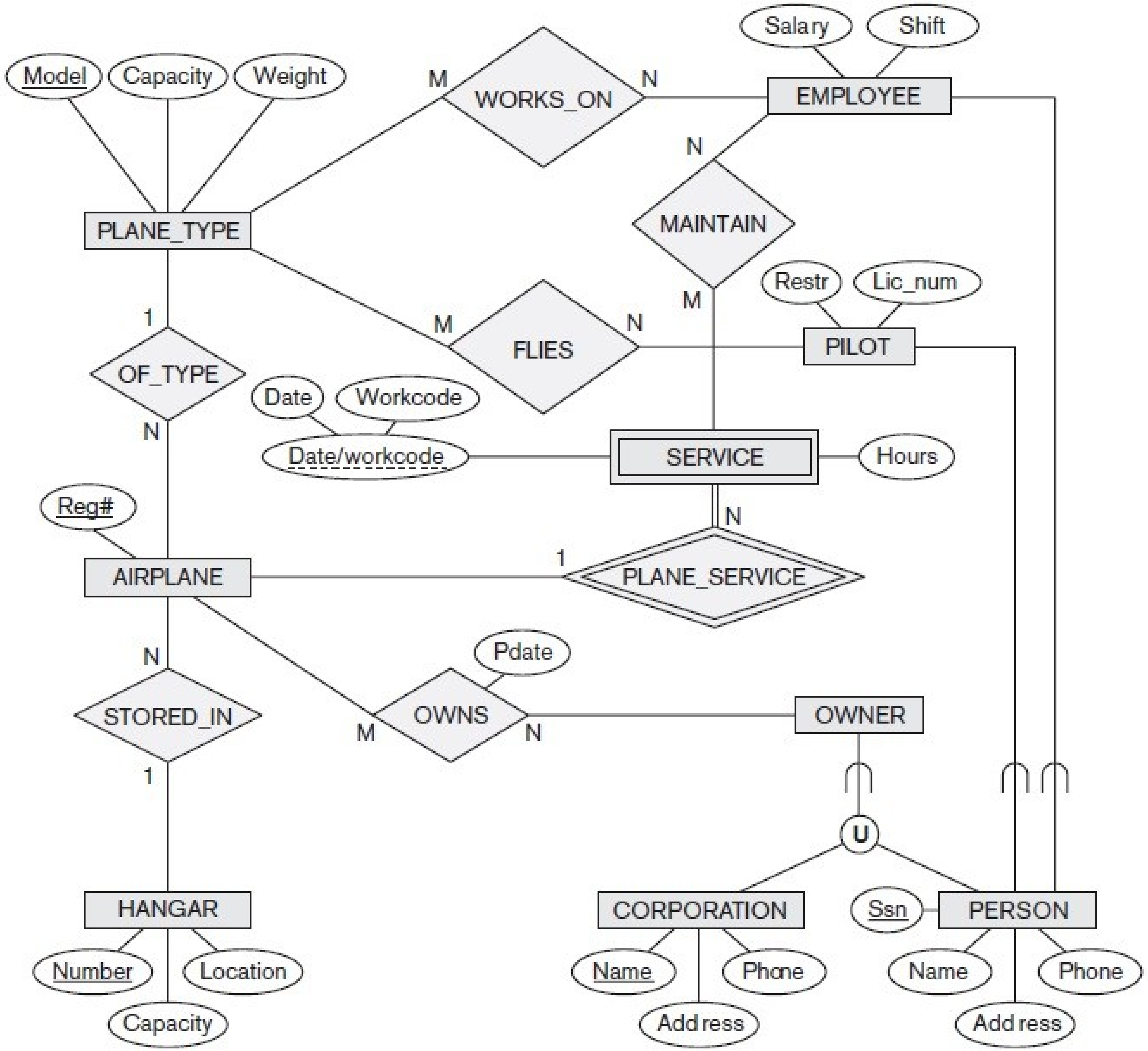
Example: 1. Map ER-Model to Relational Model


Date
Time_stamp
Time
SHIP_MOVEMENT
Longitude
N
Latitude
HISTORY
1
Type
Tonnage
Hull
Sname
SHIP
N
TYPE
1
SHIP_TYPE
Owner
(0,*)
Start_date
End_date
N
HOME_PORT
SHIP_AT_PORT
(1,1)
PORT_VISIT
Continent
Name
1
(0,*)
N
IN
1
STATE/COUNTRY
Name
Pname
PORT
N
ON
1
SEA/OCEAN/LAKE

Example: 2. Map ER-Model to Relational Model


CAR
Engine_size
Vin
Price
VEHICLE
d
TRUCK
Tonnage
Model
N
Date
SUV
No_seats
SALE
1
1
SALESPERSON
CUSTOMER
Address
City
Street
Sid
Name
Ssn
Name
State

Example: 3. Map ER-Model to Relational Model



Salary
Shift
Model
Capacity
Weight
M
WORKS_ON
N
EMPLOYEE
N
PLANE_TYPE
MAINTAIN
Restr
Lic_num
M
1
M
FLIES
N
PILOT
OF_TYPE
Date
Workcode
N
Date/workcode
SERVICE
Hours
Reg#
N
1
AIRPLANE
PLANE_SERVICE
N
Pdate
STORED_IN
OWNS
OWNER
M
N
1
U
HANGAR
CORPORATION
Ssn
PERSON
Number
Location
Name
Phone
Name
Phone
Capacity
Address
Address

https://www.youtube.com/watch?v=7LRH7DY1QbQ

DATABASE DESIGN

BITS Pilani
Pilani Campus

innovate achieve lead

BITS Pilani
Pilani Campus

SQL Commands and Relational Algebra

Learning Objectives

- SQL Commands
  - i. Definition
  - ii. Types of SQL commands

a. DDL commands b. DML Commands c. DQL commands d. DCL commands and e. TCL commands

- Setting the constraints in Create command
- Definition of Relational Algebra
- Types of Relational algebra
- Basic Operations in Relational Algebra

i. Select ii. Project iii. Union iv. Set Difference v. Cartesian product vi. Rename

SQL Commands

SQL commands are instructions coded into SQL statements which communicate with the database to perform specific tasks, work, function and queries with data.

SQL commands are classified into five categories. They are

1. DDL(Data Definition Language) Commands
2. DML (Data manipulation Language) Commands
3. DQL (Data Query Language) Commands
4. DCL (Data Control Language) Commands
5. TCL (Transaction Control Language) Commands

SQL Commands

1. **DDL commands**

DDL commands are used to make or perform changes on the physical structure of any table residing inside a database. When these commands are executed, the changes in the table are saved immediately. The DDL commands are

(i) CREATE TABLE

(ii) ALTER TABLE

(iii) DROP TABLE

(iv) RENAME

SQL Commands

(i) **CREATE TABLE-** This command is used to create the structure of the table. The syntax of this command is

```
CREATE TABLE TABLENAME ( Attribute 1 data
   type, Attribute 2 data type, …….. 
   Attribute n data type);
```

There are various data types such as integer, number, double, varchar,varchar2,date,etc.,).

Example:

```
CREATE TABLE Student
( StName varchar2(20), StNum number (3),
   Dept varchar2(20),Average number (7,2));
```

SQL Commands

The above command creates a table with four attributes as

StName	StNum	Dept	Average

(ii) ALTER TABLE- This command is used to alter the structure of the already created table such as changing size of the attribute, adding a new attribute or deleting an existing attribute. The syntax is

```
ALTER TABLE Table name Add/Modify/Drop
(specification of the alteration);
```

SQL Commands

Examples:

a. `ALTER TABLE Student Add (dob date);` // To add a new attribute dob into the Student Table

b. `ALTER TABLE Student Modify  (average number(5,2);` // Change the size of the attribute average in Student Table

c. `ALTER TABLE Student drop Dept;` // To delete a attribute Dept from the Student Table

(iii) RENAME – It is used to rename the table or database name. The syntax of the command is

```
RENAME Old table name to New table Name;
```

SQL Commands

Example:

`RENAME Student to Students;` // The table Student is renamed as Students

(iv) DROP Command – This command is used to delete a table from the database.

Its syntax is

```
DROP TABLE table name;
```

Example:

`DROP TABLE Students;` // DROP command deletes the table names Students from the database permanently

SQL Commands

2. DML commands

Once the tables are created using DDL Commands, the manipulation inside those tables is done with the DML commands. The advantages of using DML command is, if we did any wrong changes, they can be rolled back easily. The DML Commands are

(i)INSERT

(ii)DELETE

(iii)UPDATE

(iv)LOCK

SQL Commands

(i) **INSERT COMMAND** – This DML command is used to insert a tuple / record into the table. The general format of the command is

```
INSERT INTO TABLE NAME (Attribute names )
VALUES(the values of the attributes  );
```

The String and Date values are to be enclosed in Single quotes.

Examples:

a. `INSERT INTO Students VALUES (‘Yuvaraj’,10001,’CSE’,98.78);`

SQL Commands

The INSERT command inserts a new tuple into the table Students.

b. `INSERT INTO Students(StName,StNum,Average) VALUES ('Bumrah',10010,88.75);`

The above INSERT command inserts a new tuple into the table Students in which Dept is filled with NULL.After executing the above commands, the Students table will be

StName	StNum	Dept	Average
Yuvaraj	10001	CSE	95.78
Bumrah	10010	NULL	88.75

SQL Commands

(ii) DELETE COMMAND – This DML command is used to delete a tuple / record from the table. The general format of the command is

```
  DELETE *  from TABLE NAME [WHERE
Condition];
```

Here Condition is optional. If there is no condition, all the records / tuples are deleted. Otherwise, the records / tuples satisfying the condition are deleted.

Examples:

a. `DELETE *  from Students;`

The DELETE command deletes all the records in the table.

SQL Commands

a. `DELETE from Students WHERE Average<=90;`

The DELETE command deletes all the student records who have got average <= 90 from the table.

(iii) UPDATE COMMAND – This DML command is used to

Modify/change the content of a tuple / record in a table. The general format of the command is

```
UPDATE TABLE NAME SET ATT = V WHERE
     CONDITION;
```

Where

ATT is the attribute which requires modification

V is a value or an expression

SQL Commands

Examples

a. `UPDATE Student SET StName = ‘Yuvaraj Singh’WHERE StName = ‘Yuvaraj’;`

The UPDATE command modifies the StName,’Yuvaraj’ with ‘Yuvaraj Singh’

b. `UPDATE Student SET Average = Average + 2.5 WHERE StName = ‘Yuvaraj Singh’;`

The UPDATE command updates the attribute Average with Average + 2.5. That is, 2.5 marks are added with Average for all the records / tuples.

SQL Commands

(iv) LOCK Command – This command is used to lock the privilege as either READ or WRITE. Its syntax is

```
LOCK TABLE Table name READ / WRITE;
```

Example;

`LOCK TABLE Student READ` ; // Locks the privilege as READ.

3. DQL commands

DQL commands are used for performing data selection in SQL. It is performed with SELECT command. Its syntax is

```
SELECT ATTNAMES  FROM TABLE NAME WHERE
CONDITION;
```

SQL Commands

Assume that a table has the tuples as follows:

StName	StNum	Dept	Average
Yuvaraj	10001	CSE	98.78
Bumrah	10010	IT	88.75
ROY	10019	Mech	65.25
Foster	55001	CSE	50.55
BIJU	10111	IT	55.75
RAM	44448	CSE	44.00
David	12345	CSE	88.00

SQL Commands

Examples – a. `SELECT StNum,Average FROM Students;` // displays StNum and Average of all the records in Students table as

StNum	Average
10001	98.78
10010	88.75
10019	65.25
55001	50.55
101116	55.75
44448	44.00
12345	88.00

SQL Commands

b. `SELECT StNum,Average FROM Students where Average>=76;` // displays StNum and Average of all the records in Students table who have the average >=76 as

StNum	Average
10001	98.78
10010	88.75
12345	88.00

c. `SELECT * FROM Students;` // displays all the records in Students table as

SQL Commands

StName	StNum	Dept	Average
Yuvaraj	10001	CSE	98.78
Bumrah	10010	IT	88.75
ROY	10019	Mech	65.25
Foster	55001	CSE	50.55
BIJU	10111	IT	55.75
RAM	44448	CSE	44.00
David	12345	CSE	88.00

d. `SELECT * FROM Students where Average>=76;` // displays all the information of all the records in Students table who have the average >=76 as

SQL Commands

StName	StNum	Dept	Average
Yuvaraj	10001	CSE	98.78
Bumrah	10010	IT	88.75
David	12345	CSE	88.00

4. DCL commands

These command are used for granting and revoking permission to the users. The commands are GRANT and REVOKE. The syntax of GRANT command is

```
GRANT Privilege  on Object to User name;
```

Example:

```
GRANT CREATE TABLE to UserMe;
```

SQL Commands

Granting the Create command privilege to the user UserMe

The syntax of REVOKE command is

```
REVOKE Privilege  on Object from User name;
```

Example:

```
REVOKE CREATE TABLE from UserMe;
```

Revoking the Create command privilege from UserMe.

The privileges are Alter, Drop, Select, Update, Delete, etc.,

SQL Commands

5. TCL commands

These commands are applied to rollback and commit the changes what we have made in the database.

The TCL Commands are

(i)ROLLBACK – It is used to Undo the changes made in the database

(ii)Commit – It is used to save the changes made in the database.

Setting the Constraints

Integrity Constraints

We know that it is possible to set some conditions (constraints) at the time of creating the structure of the table to maintain the integrity.

Integrity checks the validity of the value given for the attribute of the table.

These constraints are called as Integrity Constraints.

When the values of the attributes are give, these constraints check whether they are valid or not.

If they are valid, the values will be permitted to construct the tuple.

Setting the Constraints

After the tuple/record is constructed, it can be added into the table.

If the value of the attribute is not valid(incorrect value) , the constraint generates the error code and is displayed on the user's screen without giving the permission to construct the tuple.

Setting the Constraints

The constraints can be set with various formats. Some of them are

Constraint Type 1. Setting an attribute as NOT NULL

This constraint is set to accept a value for mandatory attribute and it can be assigned a NULL value.

Example for setting this constraint is

```
CREATE TABLE Student
( StName varchar2(20) NOT NULL,
 StNum number NOT NULL, Dept varchar2(20),
                                            Average number (7,2));
```

Setting the Constraints

Constraint Type 2. Setting an attribute as a Primary Key.

The primary key is set for avoiding duplicate value for an attrbute.

Example for setting this constraint is

```
CREATE TABLE Student
( StName varchar2(20), StNum number NOT
  NULL, Dept varchar2(20),Average number
  (7,2) primary key(StNum));
```

Setting the Constraints

Constraint Type 3. Setting the possible value (s) for an attribute.

The set of valid values to be assigned for an attribute is given by using a constraint called as CHECK. Example2 for setting this constraint is

```
1.CREATE TABLE Student
( StName varchar2(20), StNum number NOT
  NULL, Dept varchar2(20),Average number
  (7,2) CHECK (Dept in
  (‘CSE’,’IT’,’Mech’,’BIO’))); // valid
  values for the attribute Dept is set
```

Setting the Constraints

```
2.CREATE TABLE Daily_wages
(Name varchar2(20), Num number NOT NULL,
Hrs_worked number (3,2), Wage number (4,2)
CHECK (Hrs_worked between 0 and 24)));
```

The valid value for attribute Hrs_worked is set between 0 and 24.

If the value for the attribute is given out of this range, the system generates the error message.

Relational Algebra

Relational Algebra in DBMS

Relational algebra is a **procedural** query language that performs operation on relational model.

The purpose of a query language is to retrieve data from database or perform various operations such as insert, update, delete on the data in the database.

Conversely ***relational calculus*** is a **non-procedural query language.**

It specifies what data to be retrieved from the database but doesn't specify how to retrieve it.

Relational Algebra

Types of operations in relational algebra

The types of operations in relational algebra are classified into broad two categories. They are

1. **Basic Operations** – these are the fundamental

operations which are to be performed on the data.

2. **Derived Operations** – these are the advanced operations which are used for extension operations

1. Basic/Fundamental Operations in relational algebra are:

- i. Select (σ)
- ii. Project (∏)

Relational Algebra

iii. Union (∪)

iv. Set Difference (-)

v. Cartesian product (X)

vi. Rename (ρ)

2. The Derived Operations in relational algebra are:

i. Natural Join (⋈)

ii. Left, Right, Full outer join (⟕, ⟖, ⟗)

iii. Intersection (∩)

iv. Division (÷)

BASIC OPERATIONS

i. Select Operator

Select Operator is denoted by sigma (σ) and it is used to find the tuples / rows in a relation / a table which satisfy the given condition. **Syntax of Select Operator (σ)**

σ Condition/Predicate(Relation/Table name)

Example for Select Operator

Assume that a customer table has the following data

Table: CUSTOMER

Relational Algebra - BASIC OPERATIONS

Customer_Id	Customer_Name	Customer_City
C10100	Sachin	Agra
C10111	Laxman	Agra
C10115	Shewag	Noida
C10117	Virat	Delhi
C10118	Dravid	Delhi

If the Query is

σ Customer_City="Agra" (CUSTOMER) ,

the output will be

Relational Algebra - BASIC OPERATIONS

Customer_Id	Customer_Name	Customer_City
C10100	Sachin	Agra
C10111	Laxman	Agra

Note:- The Select operation in SQL is performed with SELECT command with a where clause.

The above operation is performed with SQL command as

```
Select * from CUSTOMER where
  Customer_city=”Agra”;
```

Relational Algebra

ii. Project Operator

Project operator is denoted by ∏ symbol and it is used to select desired columns / attributes from a table / a relation.

Syntax of Project Operator

```
∏ column_name1, column_name2, ....,
  column_nameN(table_name)
```

Example for Project Operator

Consider a table CUSTOMER with three columns, we want to fetch only two columns of the table,

```
∏ Customer_Name, Customer_City (CUSTOMER)
```

The above relational algebra query selects and returns the columns specified from the table

Relational Algebra - BASIC OPERATIONS

Then the Output would be

Customer_Name	Customer_City
Sachin	Agra
Laxman	Agra
Shewag	Noida
Virat	Delhi
Dravid	Delhi

Note:- The corresponding SQL command is:

```
Select Customer_Name ,Customer_City  from
  CUSTOMER;
```

Relational Algebra - BASIC OPERATIONS

iii. Union Operator

Union operator is denoted by ∪ symbol and it is used to select all the rows / tuples from two tables / relations.

Let us discuss union operator a bit more.

Assume that there are two relations R1 and R2 both have same columns and we want to select all the tuples /rows from these relations then we can apply the union operator on these relations.

Note: The rows / tuples that are present in both the tables will only appear once in the union set.

In short we can say that there are no duplicates present after the union operation.

Relational Algebra - BASIC OPERATIONS

Syntax of Union Operator: table_name1 ∪ table_name2

Example for Union Operator

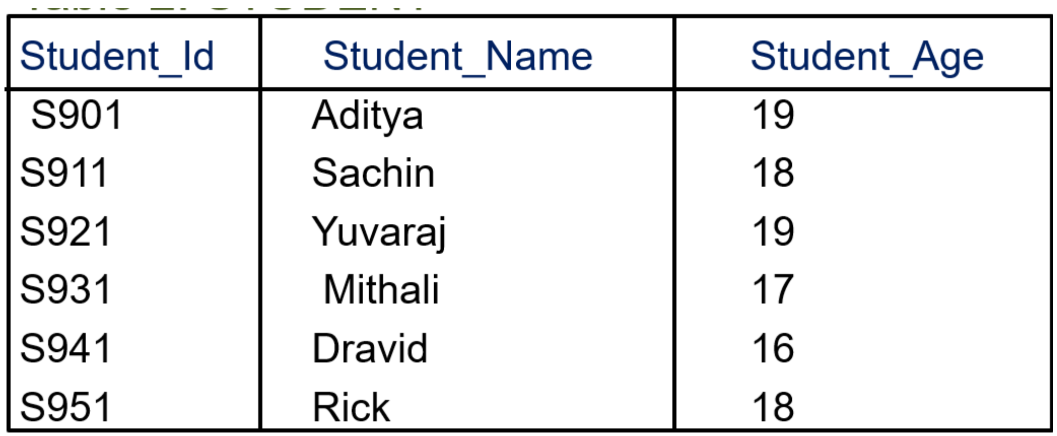
Assume that the tables COURSE and STUDENT have the data as follows

Table 1: COURSE

Course_Id	Student_Name	Student_Id
C101	Aditya	S901
C104	Aditya	S901
C106	Sachin	S911
C109	Yuvaraj	S921
C115	Mithali	S931

Relational Algebra

Table 2: STUDENT



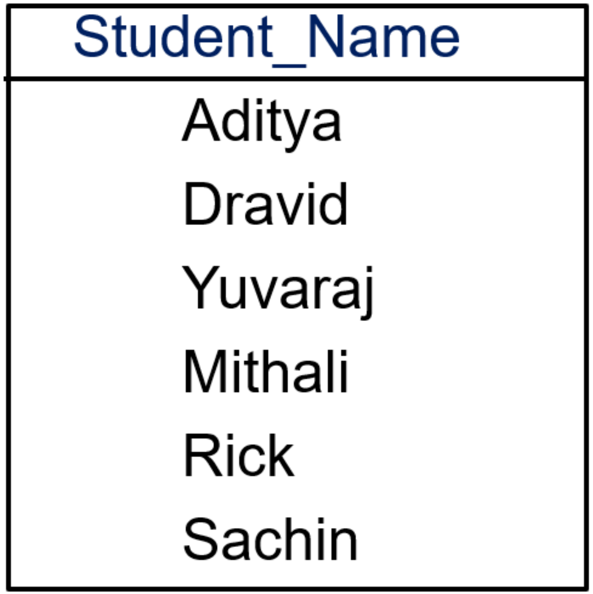
Student_Id	Student_Name	Student_Age
S901	Aditya	19
S911	Sachin	18
S921	Yuvaraj	19
S931	Mithali	17
S941	Dravid	16
S951	Rick	18

If we use the Query is executed with the intersection operator as

∏ Student_Name (COURSE) ∪ ∏ Student_Name (STUDENT),

the Output table will be

Relational Algebra



Student_Name
Aditya
Dravid
Yuvaraj
Mithali
Rick
Sachin

Note: As we can see there are no duplicate names present in the output even though we had few common names in both the tables, also in the COURSE table we had the duplicate name itself.

Relational Algebra - BASIC OPERATIONS

iv. Intersection Operator

Intersection operator is denoted by ∩ symbol and it is used to select common rows /tuples from two tables /relations.

Assume that we have two relations R1 and R2 both have same columns and we want to select all those tuples / rows that are present in both the relations, then in that case we can apply intersection operation on these two relations R1 ∩ R2.

Relational Algebra - BASIC OPERATIONS

Note: Only those rows that are present in both the tables will appear in the result set.

Syntax of Intersection Operator

`table_name1 ∩ table_name2`

Example for Intersection Operator

Let us take the same example that we have taken above operator.

If the Query is executed with the intersection operator as

`∏ Student_Name (COURSE) ∩ ∏ Student_Name (STUDENT)`, the Output table is shown as follows

Relational Algebra - BASIC OPERATIONS

Student_Name
Aditya
Sachin
Yuvaraj
Mithali

The Equivalent SQL command is

Select Student_Name from Course **INTERSECT** Select Student_Name from STUDENT;

v. Set Difference - Set Difference is denoted by – symbol.

Let us assume that we have two relations R1 and R2 and we want to select all those tuples / rows that are present in Relation R1 but **not** present in Relation R2, this can be done using Set difference R1 – R2.

Relational Algebra - BASIC OPERATIONS

Example for Set Difference

Let us take the same tables COURSE and STUDENT that we have seen above.

Query:

Let us write a query to select those student names that are present in STUDENT table but not present in COURSE table.

∏ Student_Name (STUDENT) - ∏ Student_Name (COURSE)

Relational Algebra - BASIC OPERATIONS

Student_Name
Dravid
Rick

The Equivalent SQL command is

```
Select Student_Name from STUDENT MINUS
                    Select Student_Name from COURSE;
```

Relational Algebra - BASIC OPERATIONS

vi. Cartesian product

Cartesian Product is denoted by X symbol.

Assume that R1 and R2 are the given relations.

The Cartesian product of these two relations (R1 X R2) would combine each tuple of first relation R1 with the each tuple of second relation R2.

Syntax of Cartesian product (X)

R1 X R2 ;

Relational Algebra - BASIC OPERATIONS

Example for Cartesian product

Table 1: R

Col_A	Col_B
AA	100
BB	200
CC	300

Table 2 : S

Col_X	Col_Y
XX	99
YY	11
ZZ	101

If the Query is to find the Cartesian product (RXS) of the tables R and S, the output will be

Relational Algebra - BASIC OPERATIONS

Col_A	Col_B	Col_X	Col_Y
AA	100	XX	99
AA	100	YY	11
AA	100	ZZ	101
BB	200	XX	99
BB	200	YY	11
BB	200	ZZ	101
CC	300	XX	99
CC	300	YY	11
CC	300	ZZ	101

Relational Algebra - BASIC OPERATIONS

Note: The number of rows in the output will always be the cross product of number of rows in each table.

In our example table 1 has 3 rows and table 2 has 3 rows, so the output has 3×3 = 9 rows.

vi. Rename

The rename operation is represented with the symbol ρ.

This operation is used to rename a relation or an attribute of a relation.

Rename Syntax:

ρ(new_relation_name, old_relation_name)

Relational Algebra - BASIC OPERATIONS

Example for Rename - Let us say we have a table customer, we are fetching customer names and we are renaming the resulted relation to CUST_NAMES.

Table: CUSTOMER

Customer_Id	Customer_Name	Customer_City
C10100	Sachin	Agra
C10111	Laxman	Agra
C10115	Shewag	Noida
C10117	Virat	Delhi
C10118	Dravid	Delhi

Relational Algebra - BASIC OPERATIONS

If the Query is formed as

`ρ(CUST_NAMES, ∏(Customer_Name)(CUSTOMER)),`

the output will be

CUST_NAMES
Sachin
Raghu
Shewag
Ajeet
Dravid

Thanks

DATABASE DESIGN

SESSION 6

BITS Pilani
Pilani Campus

innovate achieve lead

BITS Pilani
Pilani Campus

Relational Algebra and SQL Commands

Learning Objectives

- ➢ Derived Operations in Relational Algebra
- Different types of Joins
  - i. Inner Join
  - ii. Full Join
  - iii. Left Join
  - iv. Right Join
- ➢ Nested Query / Sub query
- ➢ SQL Aggregate Functions
- ➢ SQL EXISTS / NOT EXISTS Condition
- ➢ SQL command with GROUP By Clause
- ➢ SQL Command with HAVING clause

JOIN operation in SQL Commands

What are Joins?

JOINS in SQL are commands which are used to combine rows from two or more tables, based on an associated column between those tables. There are largely used when a user tries to extract data from tables which have one-to-many or many-to-many relationships between them.

Types of Joins

There are 4 types of Joins. They are

1. Inner Join
2. Full Join
3. Left Join
4. Right Join

JOIN operation in SQL Commands

The following represents the various Joins

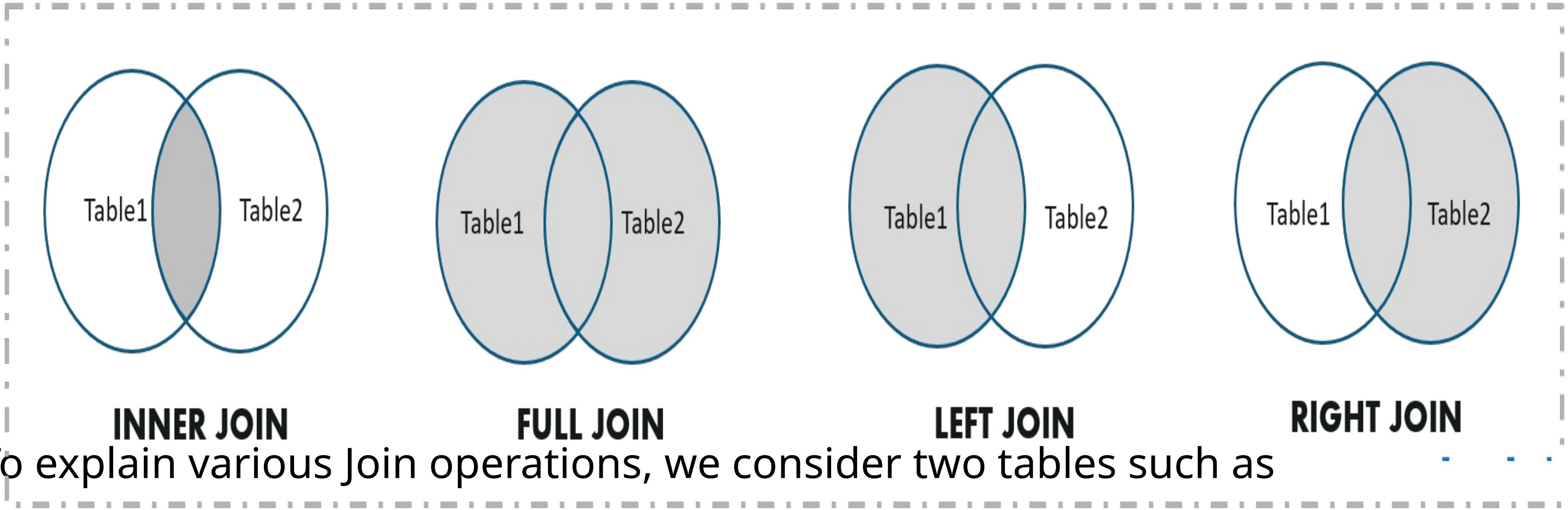


Table1
Table2
INNER JOIN
Table1
Table2
FULL JOIN
Table1
Table2
LEFT JOIN
Table1
Table2
RIGHT JOIN

To explain various Join operations, we consider two tables such as EMPLOYEE and PROJECT in which EMPLOYEE table consists of records of Employee and PROJECT contains the records of the information about various projects done by the employees

JOIN operation in SQL Commands

EMPLOYEE

EmpId	Name	Age	EmailId
E101	Mr. Dhoni	33	dhoni@abc.com
E102	Mr.Sachin	40	sachin@abc.com
E103	Mr.Rohit Sharma	30	hitman@abc.com
E104	Mr. Virat Kohli	28	virat_k@abc.com

PROJECT

ProjId	EmpId	Proj_Name	Start_date
P1	E102	Project A	07-2-2018
P2	E104	Project B	03-8-2018
P3	E102	Project C	01-1-2019
P4	E120	Project D	10-8-2019

JOIN operation in SQL Commands – Inner Join

I. INNER JOIN

INNER join returns those records which have matching values in both tables. So, if you perform an INNER join operation between the Employee table and the Projects table,

all the tuples which have matching values in both the tables will be given as output.

Syntax:

```
SELECT Table1.Column1,Table1.Column2,Table2.Column1,....
    FROM Table1INNER JOIN Table2  ON
        Table1.MatchingColumnName =
                    Table2.MatchingColumnName;
```

JOIN operation in SQL Commands – Inner Join

Example: SELECT EMPLOYEE.NAME,EMPLOYEE.EmpId, PROJECT. ProjId, PROJECT. Proj_Name FROM EMPLOYEE INNER JOIN PROJECT ON EMPLOYEE.EmpId = PROJECT. EmpId;

The output of the above query is

Name	EmpId	ProjId	Proj_Name
Mr.Sachin	E102	P1	Project A
Mr. Virat Kohli	E104	P2	Project B
Mr.Sachin	E102	P3	Project C

II. FULL JOIN

Full Join or the Full Outer Join returns all those records which either have a match in the left(Table1) or the right(Table2) table. The **Syntax** for **FULL JOIN is**

SQL Commands

```
SELECT Table1.Column1,Table1.Column2,Table2.Column1,....
   FROM  Table1 FULL JOIN  Table2 ON
          Table1.MatchingColumnName =
                     Table2.MatchingColumnName;
```

Example:

```
 SELECT EMPLOYEE.NAME,EMPLOYEE.EmpId, PROJECT. ProjId,
   PROJECT. Proj_Name FROM EMPLOYEE
     FULL JOIN PROJECT  ON
            EMPLOYEE.EmpId = PROJECT. EmpId;
```

If the above query is executed , we get the following result:

JOIN operation in SQL Commands – Left Join

Name	EmpId	ProjId	Proj_Name
Mr. Dhoni	E101	NULL	NULL
Mr.Sachin	E102	P1	Project A
Mr.Sachin	E102	P3	Project C
Mr.Rohit Sharma	E103	NULL	NULL
Mr. Virat Kohli	E104	P2	Project B
NULL	NULL	P4	Project D

III. LEFT JOIN

The LEFT JOIN or the LEFT OUTER JOIN returns all the records from the left table and also those records which satisfy a condition from the right table. Also, for the records having no matching values in the right table, the output or the result-set

JOIN operation in SQL Commands – Left Join

will contain the NULL values.

Its Syntax:

```
SELECT Table1.Column1,Table1.Column2,Table2.Column1,....
      FROM Table1 LEFT  JOIN Table2  ON
           Table1.MatchingColumnName =
                          Table2.MatchingColumnName;
```

Example:

```
SELECT EMPLOYEE.NAME,EMPLOYEE.EmpId, PROJECT. ProjId,
    PROJECT. Proj_Name FROM EMPLOYEE
       LEFT  JOIN PROJECT  ON
                 EMPLOYEE.EmpId = PROJECT. EmpId;
```

If the above query is executed , we get the following result:

JOIN operation in SQL Commands – Right Join

EmpName	EmpId	ProjId	Proj_Name
Mr. Dhoni	E101	NULL	NULL
Mr.Sachin	E102	P1	Project A
Mr.Sachin	E102	P3	Project C
Mr.Rohit Sharma	E103	NULL	NULL
Mr. Virat Kohli	E104	P2	Project B

III. RIGHT JOIN

The RIGHT JOIN or the RIGHT OUTER JOIN returns all the records from the right table and also those records which satisfy a condition from the left table. Also, for the records having no matching values in the left table, the output or the result-set will contain the NULL values.

JOIN operation in SQL Commands – Right Join

Its Syntax:

```
SELECT Table1.Column1,Table1.Column2,Table2.Column1,....
      FROM Table1 RIGHT  JOIN Table2  ON
           Table1.MatchingColumnName =
                          Table2.MatchingColumnName;
```

Example:

```
SELECT EMPLOYEE.NAME,EMPLOYEE.EmpId, PROJECT. ProjId,
    PROJECT. Proj_Name FROM EMPLOYEE
       RIGHT  JOIN PROJECT  ON
                 EMPLOYEE.EmpId = PROJECT. EmpId;
```

If the above query is executed , we get the following result:

JOIN operation in SQL Commands – Right Join

EmpName	EmpId	ProjId	Proj_Name
Mr.Sachin	E102	P1	Project A
Mr. Virat Kohli	E104	P2	Project B
Mr.Sachin	E102	P3	Project C
NULL	NULL	P4	Project D

SQL NESTED QUERY

SUBQUERY / NESTED QUERY

A subquery consists of two queries as out and inner queries.

A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.

There are a few rules that subqueries must satisfy -

- Subqueries must be enclosed within parentheses.
- A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.

SQL NESTED QUERY

- An ORDER BY command cannot be used in a subquery,

although the main query can use an ORDER BY. The GROUP BY command can be used to perform the same function as the ORDER BY in a subquery.

- Subqueries that return more than one row can only be used

with multiple value operators such as the IN operator.

- A subquery cannot be immediately enclosed in a set function.

1. Subqueries with the SELECT Statement

Subqueries are most frequently used with the SELECT

statement. The syntax for a subquery is as follows -

SQL NESTED QUERY

```
SELECT column_name [, column_name ]  FROM   table1 [, table2 ]
     WHERE  column_name   OPERATOR
   (SELECT column_name [, column_name ]
 FROM table1 [, table2 ]    [WHERE  condition]);
```

Example

Consider the CUSTOMERS table having the following records −

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	35	Ahmedabad	2000.00
2	Keshav	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitanya	25	Mumbai	6500.00
5	Rajesh	27	Bhopal	8500.00
6	Kamal	22	MP	4500.00

SQL NESTED QUERY

Now, let us check the following subquery with a SELECT statement.

```
SQL> SELECT *  FROM CUSTOMERS   WHERE ID IN (SELECT ID FROM
    CUSTOMERS  WHERE SALARY > 4500) ;
```

This would produce the following result.

ID	NAME	AGE	ADDRESS	SALARY
4	Chaitanya	25	Mumbai	6500.00
5	Rajesh	27	Bhopal	8500.00

2. Subqueries with the INSERT Statement

Subqueries also can be used with INSERT statements. The INSERT statement uses the data returned from the subquery to insert into another table. The selected data in the subquery

SQL NESTED QUERY

can be modified with any of the character, date or number functions.The basic syntax is as follows.

```
INSERT INTO table_name [ (column1 [, column2 ]) ]
        SELECT [ *|column1 [, column2 ]
               FROM table1 [, table2 ]
                            [ WHERE VALUE OPERATOR ]
```

Example

Assume a table CUSTOMER_TEMP has similar structure as CUSTOMER table. To copy the complete CUSTOMER table into the CUSTOMER_TEMP table, we can use the following SQL subquery command

SQL NESTED QUERY

```
INSERT INTO CUSTOMER_TEMP
        SELECT * FROM CUSTOMER  WHERE ID IN
                (SELECT ID  FROM CUSTOMER) ;
```

As usual, the inner query is executed first. When it is executed, it creates a table with all IDs from the CUSTOMER table. Then all the records of the CUSTOMER matching the condition are retrieved (in this case, all the records satisfy the condition) and are inserted into the table CUSTOMER_TEMP.

3. Subqueries with the UPDATE Statement

The subquery can be used in conjunction with the UPDATE statement. Either single or multiple columns in a table can be updated when using a subquery with the UPDATE statement.

SQL NESTED QUERY

The basic syntax is as follows.

```
UPDATE table
SET column_name = new_value [ WHERE OPERATOR
    [ VALUE ]     (SELECT COLUMN_NAME   FROM
                 TABLE_NAME)  [ WHERE condition) ]
```

Example

Assume that we have CUSTOMER_TEMP table available which is backup of CUSTOMER table. The following example Increments the SALARY by 10 % in the CUSTOMERS table for all the customers whose AGE is greater than or equal to 27.

SQL NESTED QUERY

```
UPDATE CUSTOMER
        SET SALARY = SALARY + SALARY * 0.10
            WHERE AGE IN
       (SELECT AGE FROM CUSTOMER_TEMP
                        WHERE AGE >= 27 );
```

The inner query is executed first which retrieve all the ages of the records in the CUSTOMER table satisfying the condition (>=27). Then the outer query is performed to update the salary for the records. The resultant CUSTOMER table is

SQL NESTED QUERY

CUSTOMER

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	35	Ahmedabad	2200.00
2	Keshav	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitanya	25	Mumbai	6500.00
5	Rajesh	27	Bhopal	9350.00
6	Kamal	22	MP	4500.00

4. Subqueries with the DELETE Statement

The subquery can be used in conjunction with the DELETE statement like with any other statements mentioned above.

SQL NESTED QUERY

Its syntax is :

```
DELETE FROM TABLE_NAME [ WHERE OPERATOR [ VALUE ]
      (SELECT COLUMN_NAME
                FROM TABLE_NAME)    [ WHERE Condition) ]
```

Example

Assuming, we have a CUSTOMER_TEMP table available which is a backup of the CUSTOMER table. The following example deletes the records from the CUSTOMER table for all the customers whose AGE is greater than or equal to 25.

```
DELETE FROM CUSTOMER
   WHERE AGE IN (SELECT AGE FROM CUSTOMER_TEMP
      WHERE AGE >= 25 );
```

SQL NESTED QUERY

Now the CUSTOMER table is

CUSTOMER

ID	NAME	AGE	ADDRESS	SALARY
3	kaushik	23	Kota	2000.00
6	Kamal	22	MP	4500.00

SQL Aggregate Functions

AGGREGATE FUNCTIONS

The aggregate functions are the predefined functions to calculate average, sum and count the number of occurrences of the records satisfying a cretria . The aggregate functions are SQL COUNT(), AVG() and SUM() Functions

1.COUNT() function – It returns the number of rows that Matches a specified criterion.

Its syntax is

```
SELECT COUNT(column_name) FROM table_name
        WHERE condition;
```

2.**AVG()** function – It returns the average value of a numeric column. Its syntax is

SQL Aggregate Functions

```
SELECT AVG(column_name)
       FROM table_name
                    WHERE condition;
```

3. **SUM()** function - It returns the total sum of a numeric column. Its syntax is

```
SELECT SUM(column_name)
    FROM table_name
                  WHERE condition;
```

Examples for the aggregate functions:

Assume that we are given a CUSTOMER table as follows:

SQL Aggregate Functions

CUSTOMER

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	35	Ahmedabad	2200.00
2	Keshav	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitanya	25	Mumbai	6500.00
5	Rajesh	27	Bhopal	9800.00

Example 1: If we execute the SQL command

SELECT COUNT(ID) FROM CUSTOMER
WHERE age >=25; , the output is

COUNT(ID)
4

SQL Aggregate Functions

Example 2: If we execute the SQL command as

```
SELECT SUM(salary)
  FROM CUSTOMER
        WHERE age >=25;
```

, the output is

SUM(salary)
20000

Example 3: If we execute the SQL command as

```
SELECT AVG(salary)
  FROM CUSTOMER
        WHERE age >=25;
```

, the output is

AVG(salary)
20000

SQL Exists / Not Exists Condition

SQL EXISTS

The EXISTS condition in SQL is used to check whether the result of a correlated nested query is empty (contains no tuples) or not. The result of EXISTS is a boolean value which is either True or False. This condition can be used in a SELECT, UPDATE, INSERT or DELETE statement.

Its Syntax is:

```
SELECT column_name(s) FROM table_name WHERE
  EXISTS
              (SELECT column_name(s)  FROM table_name
                                             WHERE condition);
```

SQL Exists / Not Exists Condition

Example: Consider the following two relations “Customer” and “Order”.

CUSTOMER

CustId	CustName	Location
C001	Mr. Ram	Chennai – 10
C008	Ms.Geetha	Chennai-22
C110	Mr.Anand	Chennai-78

ORDER

OrderId	CustId	Amount
O1001	C008	12345
O1002	C001	50000
O998	C222	43000

SQL Exists / Not Exists Condition

Using EXISTS condition with SELECT statement

To fetch the name of the customers who placed atleast one order.

```
SELECT CustName   FROM Customer WHERE
EXISTS (SELECT *   FROM Order  WHERE
            Customer. CustId = Order. CustId);
```

The output is

CustName
Ms.Geetha
Mr. Ram

SQL Exists / Not Exists Condition

Using NOT with EXISTS

To fetch name of the customers who has not placed any order.

```
SELECT CustName   FROM Customer WHERE
  NOT EXISTS (SELECT *   FROM Order  WHERE
        Customer. CustId = Order. CustId);
```

CustName
Mr.Anand

SQL Command with Group By option

The GROUP BY clause is used in the SELECT statement .

Optionally it is used in conjunction with aggregate functions to produce summary reports from the database.

That's what it does, summarizing data from the database.

The queries that contain the GROUP BY clause are called grouped queries and only return a single row for every grouped item.

GROUP BY Syntax

Now that we know what the GROUP By clause is, let's look at the syntax for a basic group by query.

SELECT statement... GROUP BY column_name1[,column_name2,...] [HAVING condition];

SQL Command with Group By option

Where

(i) "SELECT statement..." is the standard SQL SELECT statement

(ii) "GROUP BY column_name1" is the clause that performs the grouping based on a particular column which is specified as column_name1.

(iii) "[,column_name2,...]" is optional; represents other column names when the grouping is done on more than one columns.

(iv) "[HAVING condition]" is optional; it is used to restrict the rows affected by the GROUP BY clause. It is similar to the WHERE clause.

SQL Command with Group By option

Grouping using a Single Column

Assume that a table **members** has the data as follows:

Name	**MemNo**	Gender	Age	Place
Ram	M1001	Male	25	Chennai-1
Hema	M1010	Female	23	Chennai-10
Amrutha	M1005	Female	19	Chennai-5
Akshitha	M1111	Female	24	Chennai-45
Anand	M0009	Male	45	Chennai-78

In order to help understand the effect of Group By clause, let us execute a simple query that returns all the gender entries from the members table.

SELECT Gender FROM members ;

SQL Command with Group By option

It produces the following output:

Gender
Male
Female
Female
Female
Male

Suppose we want to get the unique values for genders. We can use a following query -

SELECT Gender FROM members GROUP BY Gender;

It produces the following output:

SQL Command with Group By option

Gender
Male
Female

Note that it returns only two results. This is because we only have two gender types **Male** and **Female**. The GROUP BY clause grouped all the "Male" members together and returned only a single row for it. It is same with the "Female" members.

If we use the SQL Query as

```
SELECT  Gender,Avg(Age)  FROM members GROUP BY Gender;
```

it returns the average Age value of each Gender as

SQL Command with Group By option

Gender	Avg(Age)
Male	35
Female	22

(25+45) / 2

(23+19+24)/3

Grouping using a Multiple Columns

Assume that a table **Orders** has the data as follows:

SalesId	**CustId**	Amount
S1000	C5001	8000
S1003	C5003	13000
S1000	C5010	28000
S1000	C5004	40000
S1000	C5001	70000
S1003	C5001	8000

SQL Command with Group By option

SELECT SalesId,CustId,Sum(Amount),Avg(Amount) FROM Orders GROUP BY SalesId, CustId;

it displays the output as

SalesId	**CustId**	Sum(Amount)	Avg(Amount)
S1000	C5001	78000	39000
S1000	C5010	28000	28000
S1000	C5004	40000	40000
S1003	C5003	13000	13000
S1003	C5001	8000	8000

In the above query, the records are grouped with two columns

/ attributes such as SalesId and CustId.

SQL Command with Group By option

If the same table is grouped with a single attribute as

SELECT SalesId,Sum(Amount),Average(Amount) FROM Orders GROUP BY SalesId;

it displays the output as

SalesId	Sum(Amount)	Average(Amount)
S1000	146000	36500
S1003	21000	10500

SQL Command with Having option

We know that WHERE clause is used to select and reject the individual rows from a table, the HAVING is also used to select and reject row groups. The format of the HAVING clause parallels that of the WHERE clause, consisting of the keyword HAVING followed by the search condition. Thus, the HAVING clause specifies a search condition for groups.

If a query is asked as , “What is the average salesperson whose order total more than Rs 30,000?”, we can use the HAVING clause as

SELECT SalesId,Sum(Amount),Average(Amount) FROM Orders GROUP BY SalesId HAVING Sum(Amount)>30000;

SQL Command with Having option

it displays the output as

SalesId	Sum(Amount)	Average(Amount)
S1000	146000	36500

Select Statement with Order By Clause

The Order By clause in the Select statement is used to sort the query result in an order of a column. The order may be in ascending order or descending order. The default order is ascending.

Assume that a table Student has the following data

StName	Stno	Dept	Percent
Dravid	S110	Mech	78
Meera	S200	Cse	92
Levin	S078	Mech	80
Leela	S012	Mech	62
Raman	S003	Mech	91
Pavithra	S235	Cse	77
Pavithra	S234	Cse	67

Select Statement with Order By Clause

If we want to sort the records of the Student table according to

the department, we use the Order by clause as

Select * from Student order by Dept;

The output of the above query is

StName	**Stno**	Dept	Percent
Meera	S200	Cse	92
Pavithra	S235	Cse	77
Pavithra	S234	Cse	67
Levin	S078	Mech	80
Leela	S012	Mech	62
Raman	S003	Mech	91
Dravid	S110	Mech	78

Select Statement with Order By Clause

If we want to sort the records of the Student table according to the department and Stno within the department, we use the Order by clause as **Select * from Student order by Dept,Stno;** The output of the above query is

StName	**Stno**	Dept	Percent
Meera	S200	Cse	92
Pavithra	S234	Cse	67
Pavithra	S235	Cse	77
Raman	S003	Mech	91
Leela	S012	Mech	62
Levin	S078	Mech	80
Dravid	S110	Mech	78

Select Statement with Order By Clause

If we want to sort the Student table according to the descending values of percentage of the students,we use the SQL statement as **Select * from Student order by Percent desc;** The output of the above query is

StName	**Stno**	Dept	Percent
Meera	S200	Cse	92
Raman	S003	Mech	91
Levin	S078	Mech	80
Dravid	S110	Mech	78
Pavithra	S235	Cse	77
Pavithra	S234	Cse	67
Leela	S012	Mech	62

Thanks

Database Design

SESSION 7

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Contact Session 7:

Session Outline

- Exercises in Nested Query
- Features of Correlated Sub queries
- Trigger
- Views
- Substring handling and other features of SQL

Some More Examples in Nested Queries, Tuples, and Set/Multiset Comparisons

- ❑ Some queries require that existing values in the database be fetched and then used in a comparison condition.

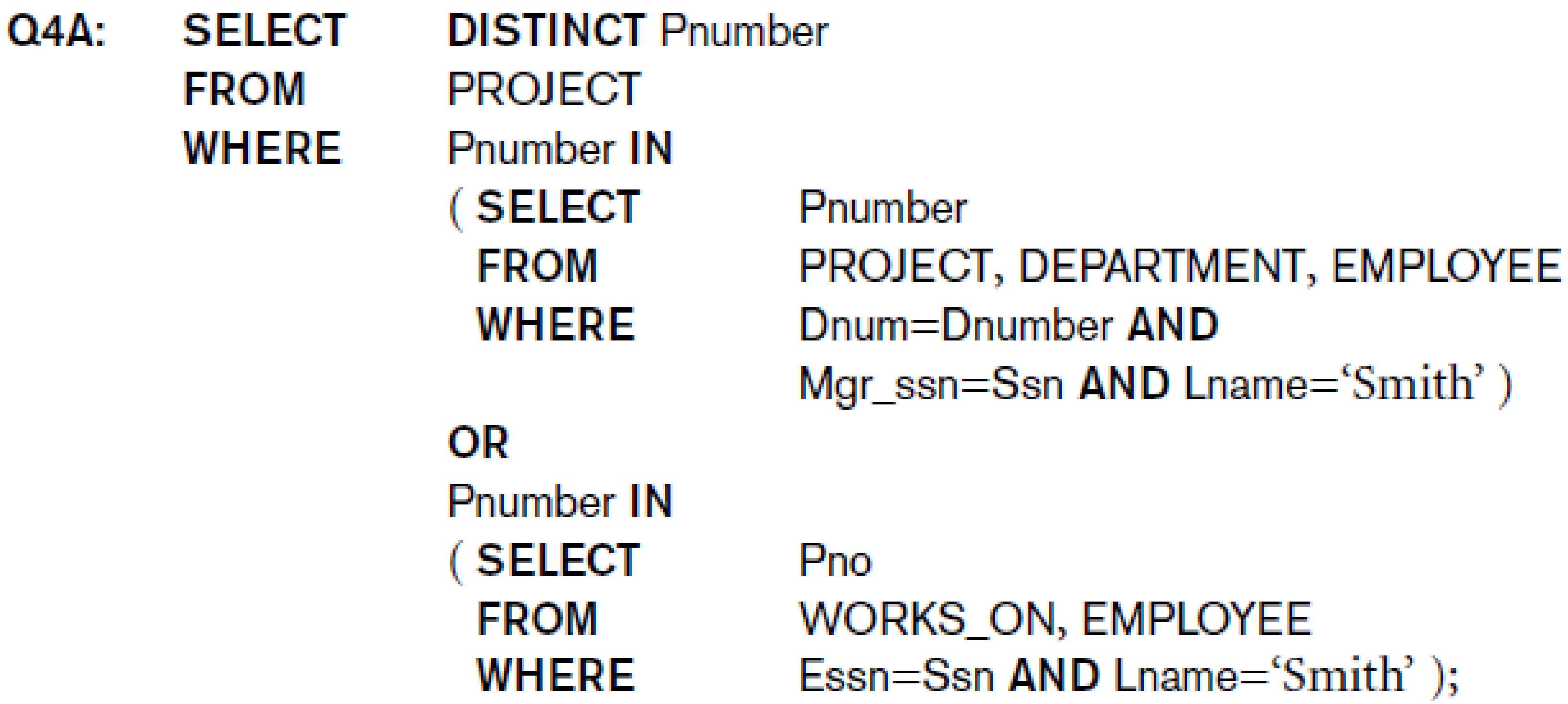
- ❑ Such queries can be conveniently formulated by using nested queries, which are complete select-from-where blocks within another SQL query.

- ❑ That other query is called the outer query.

- ❑ These nested queries can also appear in the WHERE clause or the FROM clause or the SELECT clause or other SQL clauses as needed.

Nested Queries - Example

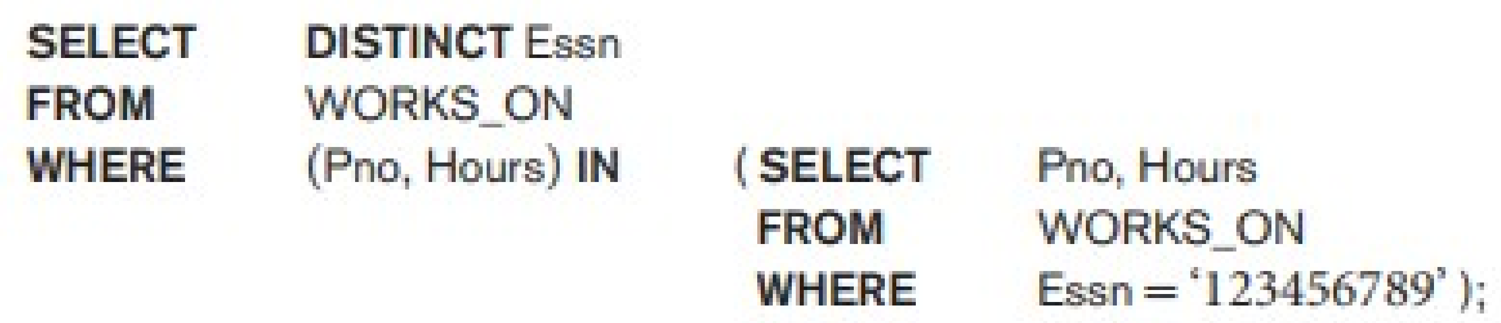
A Nested query for make a list of all project numbers for projects that involve employee Smith either as worker or as a manager of the department that controls the project:



```
Q4A:    SELECT      DISTINCT Pnumber
        FROM        PROJECT
        WHERE       Pnumber IN
                    ( SELECT       Pnumber
                      FROM         PROJECT, DEPARTMENT, EMPLOYEE
                      WHERE        Dnum=Dnumber AND
                                   Mgr_ssn=Ssn AND Lname='Smith' )
                    OR
                    Pnumber IN
                    ( SELECT       Pno
                      FROM         WORKS_ON, EMPLOYEE
                      WHERE        Essn=Ssn AND Lname='Smith' );
```

Nested Queries - Example

- If a nested query returns a single attribute and a single tuple, the query result will be a single (scalar) value. In such cases, it is permissible to use = instead of IN for the comparison operator.
- To illustrate this, consider the following query:



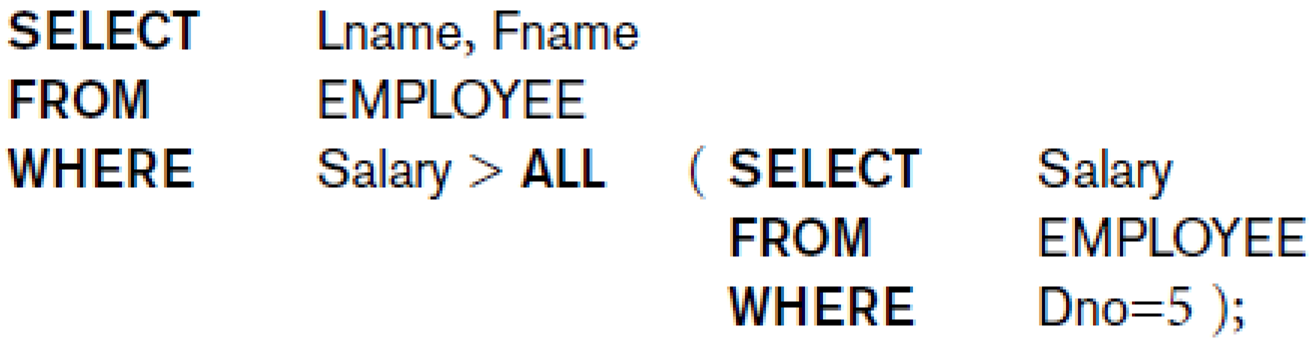
```
SELECT    DISTINCT Essn
FROM      WORKS_ON
WHERE     (Pno, Hours) IN    ( SELECT    Pno, Hours
                               FROM      WORKS_ON
                               WHERE     Essn = '123456789' );
```

This query will select the Essns of all employees who work the same (project, hours) combination on some project that employee 'John Smith' (whose Ssn = '123456789') works on.

Nested Queries - Example

Use other comparison operators to compare a single value *v*

- = ANY (or = SOME) operator [equivalent to IN]
    - Returns TRUE if the value *v* is equal to some value in the set
- Other operators that can be combined with ANY (or SOME): >, >=, <, <=, and <>
- ALL: value must exceed all values from nested query



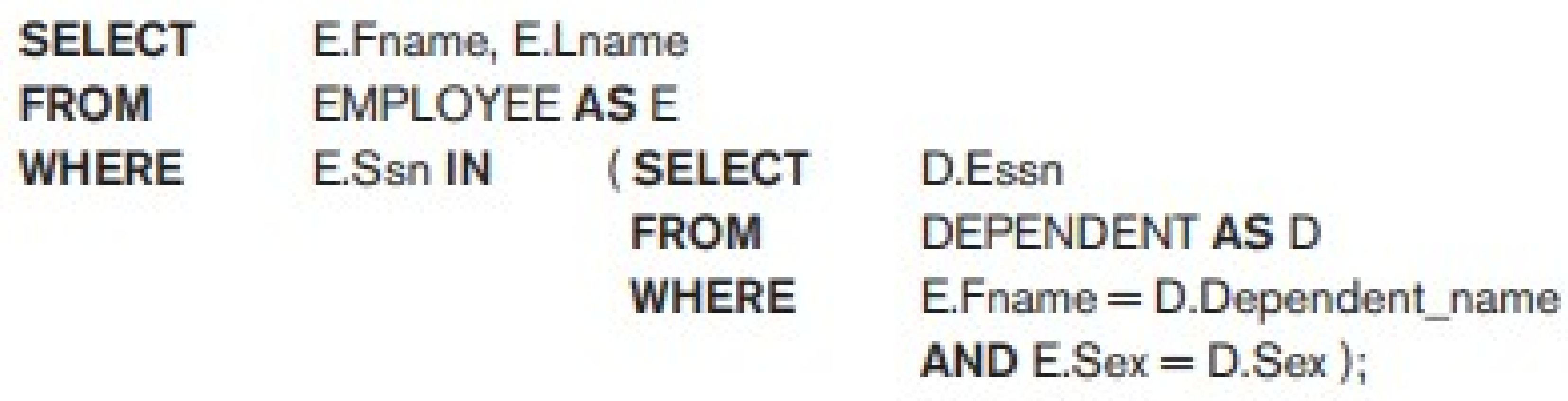
```
SELECT    Lname, Fname
FROM      EMPLOYEE
WHERE     Salary > ALL    ( SELECT    Salary
                            FROM      EMPLOYEE
                            WHERE     Dno=5 );
```

Above query returns the names of employees whose salary is greater than the salary of all the employees in department 5:

Nested Queries - Example

A reference to an unqualified attribute refers to the relation declared in the innermost nested query.

Example: Retrieve the name of each employee who has a dependent with the same first name and is the same sex as the employee.



```
SELECT    E.Fname, E.Lname
FROM      EMPLOYEE AS E
WHERE     E.Ssn IN    ( SELECT    D.Essn
                        FROM      DEPENDENT AS D
                        WHERE     E.Fname = D.Dependent_name
                                  AND E.Sex = D.Sex );
```

In above query, we must qualify E.Sex because it refers to the Sex attribute of EMPLOYEE from the outer query, and DEPENDENT also has an attribute called Sex. If there were any unqualified references to Sex in the nested query, they would refer to the Sex attribute of DEPENDENT.

1. Correlated Nested Queries

- Whenever a condition in the WHERE clause of a nested query references some attribute of a relation declared in the outer query, the two queries are said to be correlated
- Queries that are nested using the = or IN comparison operator can be collapsed into one single block: Example:
- Example 1.3A:
- **Query:**

```
SELECT   E.Fname, E.Lname
FROM     EMPLOYEE AS E, DEPENDENT AS D
WHERE    E.Ssn=D.Essn AND E.Sex=D.Sex AND
         E.Fname=D.Dependent_name;
```

- **Correlated** nested query
  - Evaluated once for each tuple in the outer query

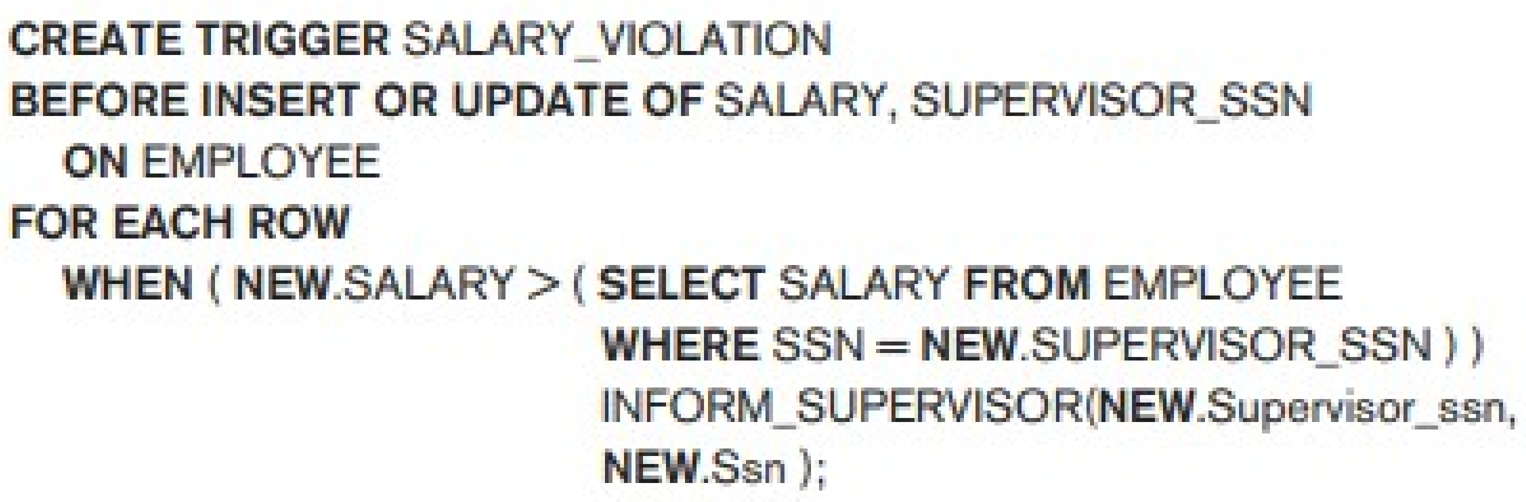
2. Introduction to Triggers in SQL

- CREATE TRIGGER statement
  - Used to monitor the database
- Typical trigger has three components which make it a rule for an “active database”.
  - **Event(s)**
  - **Condition**
  - **Action**

Introduction to Triggers: USE OF TRIGGERS

- AN EXAMPLE with standard Syntax.
- Suppose we want to check whenever an employee’s salary is greater than the salary of his or her direct supervisor in the COMPANY database. Several events can trigger this rule: inserting a new employee record, changing an employee’s salary, or changing an employee’s supervisor. Suppose that the action to take would be to call an external stored procedure SALARY_VIOLATION, which will notify the supervisor

Introduction to Triggers: USE OF TRIGGERS



```
CREATE TRIGGER SALARY_VIOLATION
BEFORE INSERT OR UPDATE OF SALARY, SUPERVISOR_SSN
   ON EMPLOYEE
FOR EACH ROW
   WHEN ( NEW.SALARY > ( SELECT SALARY FROM EMPLOYEE
                          WHERE SSN = NEW.SUPERVISOR_SSN ) )
                          INFORM_SUPERVISOR(NEW.Supervisor_ssn,
                          NEW.Ssn );
```

3. Views (Virtual Tables) in SQL

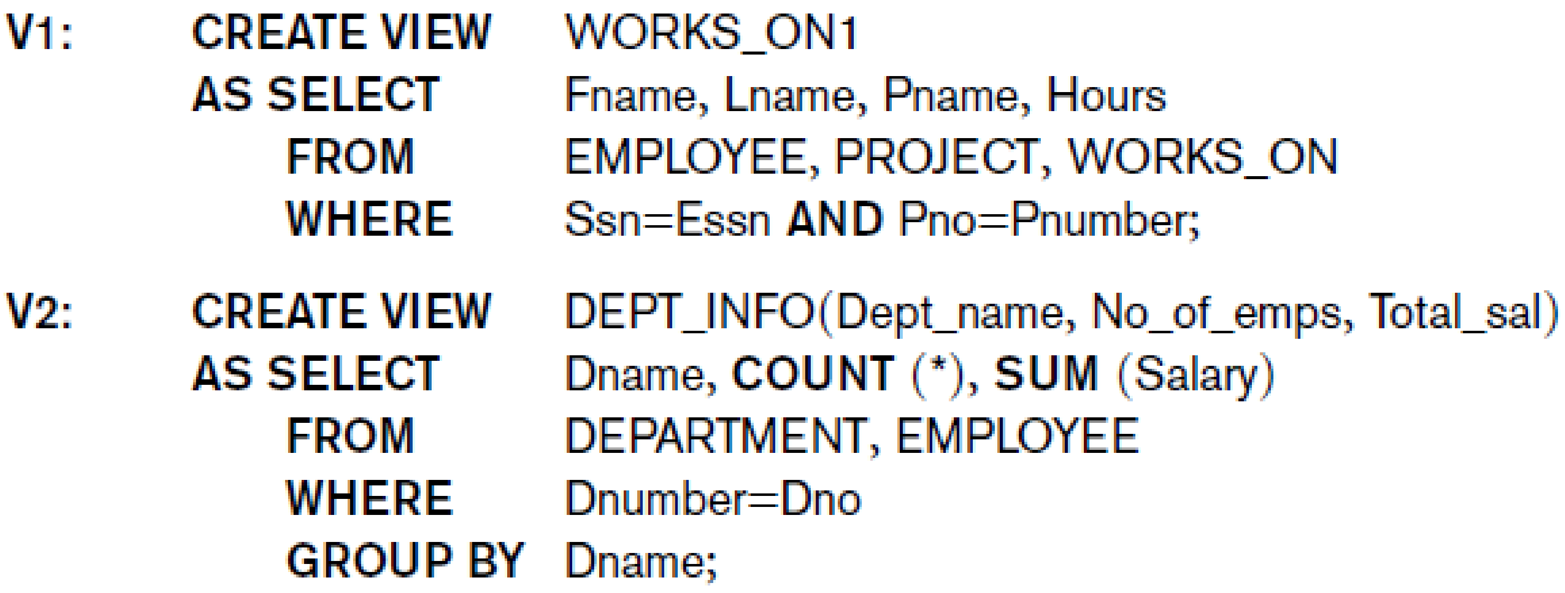
- Concept of a view in SQL
- Specification of Views in SQL
- View Implementation, View Update, and Inline Views
- Views as Authorization Mechanisms

Concept of a view in SQL

- Concept of a view in SQL:
  - Single table derived from other tables called the **defining tables**
  - Considered to be a virtual table that is not necessarily populated

Specification of Views in SQL

- `CREATE VIEW` command
  - Give table name, list of attribute names, and a query to specify the contents of the view
  - In V1, attributes retain the names from base tables and In V2, attributes are assigned names



```
V1:     CREATE VIEW    WORKS_ON1
        AS SELECT      Fname, Lname, Pname, Hours
            FROM       EMPLOYEE, PROJECT, WORKS_ON
            WHERE      Ssn=Essn AND Pno=Pnumber;

V2:     CREATE VIEW    DEPT_INFO(Dept_name, No_of_emps, Total_sal)
        AS SELECT      Dname, COUNT (*), SUM (Salary)
            FROM       DEPARTMENT, EMPLOYEE
            WHERE      Dnumber=Dno
            GROUP BY   Dname;
```

Specification of Views in SQL

- Once a View is defined, SQL queries can use the View relation in the FROM clause
- View is always up-to-date
  - Responsibility of the DBMS and not the user
- **DROP VIEW** command
  - Dispose of a view

View Implementation, View Update, and Inline Views

- Complex problem of efficiently implementing a view for querying.

- **Strategy1: Query modification** approach
  - Compute the view as and when needed. Do not store permanently
  - Modify view query into a query on underlying base tables
  - Disadvantage: inefficient for views defined via complex queries that are time-consuming to execute

View Implementation, View Update, and Inline Views

Strategy 2: View materialization

- Physically create a temporary view table when the view is first queried.
- Keep that table on the assumption that other queries on the view will follow.
- Requires efficient strategy for automatically updating the view table when the base tables are updated.

Incremental update strategy for materialized views

- DBMS determines what new tuples must be inserted, deleted, or modified in a materialized view table.

View Implementation, View Update, and Inline Views

- Multiple ways to handle materialization:

I. Immediate update strategy updates a view as soon as the base tables are changed

II. Lazy update strategy updates the view when needed by a view query

III. Periodic update strategy updates the view periodically (in the latter strategy, a view query may get a result that is not up-to-date). This is commonly used in Banks, Retail store operations, etc.

View Update

Update on a view defined on a single table without any aggregate functions

- Can be mapped to an update on underlying base table- possible if the primary key is preserved in the view

Update not permitted on aggregate views. E.g.,

```
UPDATE          DEPT_INFO
SET        Total_sal=100000
WHERE      Dname='Research';
```

Cannot be processed because Total_sal is a computed value in the view definition

Views as authorization mechanism

SQL query authorization statements Views can be used to hide certain attributes or tuples from unauthorized users.

Suppose a certain user is only allowed to see employee information for employees who

work for department 5;

Then we can create the following view DEPT5EMP and grant the user the privilege to query the view but not the base table EMPLOYEE itself.

This user will only be able to retrieve employee information for employee tuples whose Dno = 5, and will not be able to see other employee tuples when the view is queried.

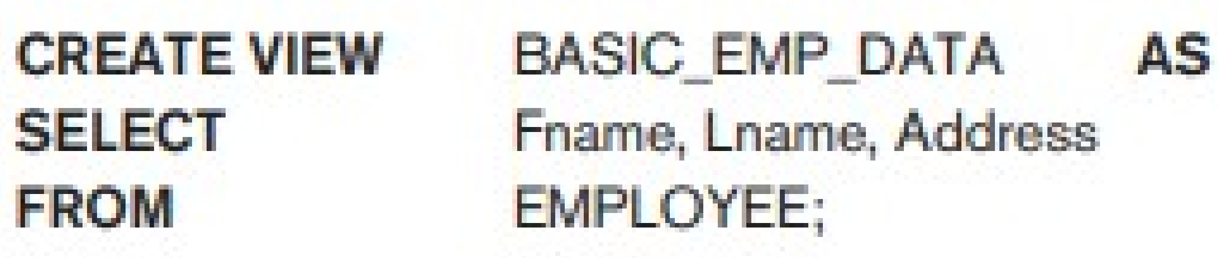
Views as authorization mechanism

```
CREATE VIEW         DEPT5EMP   AS
SELECT              *
FROM         EMPLOYEE
WHERE               Dno = 5;
```

In a similar manner, a view can restrict a user to only see certain columns; for

example, only the first name, last name, and address of an employee may be visible

as follows:



```
CREATE VIEW     BASIC_EMP_DATA      AS
SELECT          Fname, Lname, Address
FROM            EMPLOYEE;
```

Summary of SQL Syntax

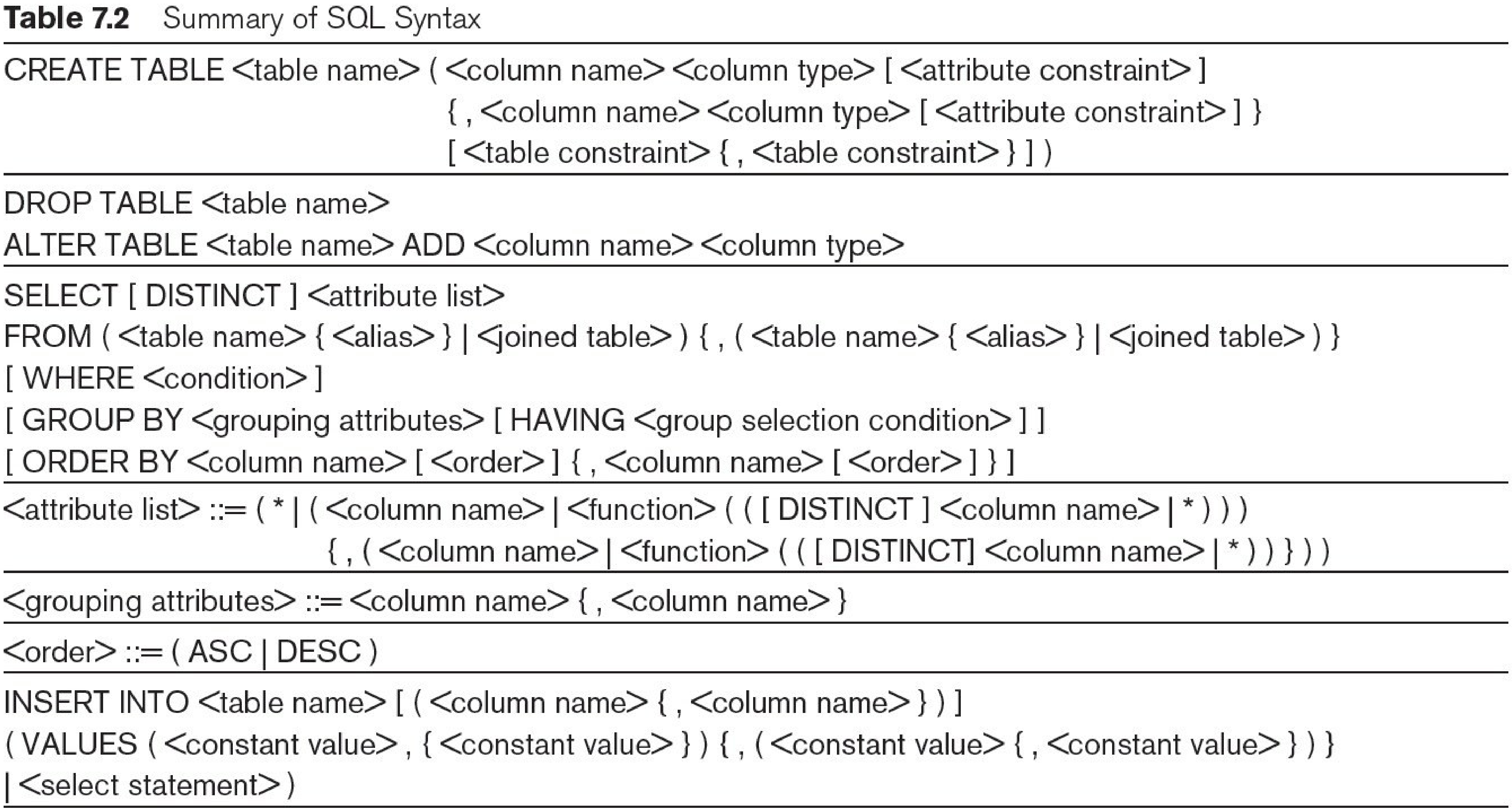


Table 7.2 Summary of SQL Syntax

```
CREATE TABLE <table name> ( <column name> <column type> [ <attribute constraint> ]
                           { , <column name> <column type> [ <attribute constraint> ] }
                           [ <table constraint> { , <table constraint> } ] )
```

```
DROP TABLE <table name>
ALTER TABLE <table name> ADD <column name> <column type>
```

```
SELECT [ DISTINCT ] <attribute list>
FROM ( <table name> { <alias> } | <joined table> ) { , ( <table name> { <alias> } | <joined table> ) }
[ WHERE <condition> ]
[ GROUP BY <grouping attributes> [ HAVING <group selection condition> ] ]
[ ORDER BY <column name> [ <order> ] { , <column name> [ <order> ] } ]
```

```
<attribute list> ::= ( * | ( <column name> | <function> ( ( [ DISTINCT ] <column name> | * ) ) )
                    { , ( <column name> | <function> ( ( [ DISTINCT] <column name> | * ) ) } ) )
```

```
<grouping attributes> ::= <column name> { , <column name> }
```

```
<order> ::= ( ASC | DESC )
```

```
INSERT INTO <table name> [ ( <column name> { , <column name> } ) ]
( VALUES ( <constant value> , { <constant value> } ) { , ( <constant value> { , <constant value> } ) }
| <select statement> )
```

Copyright @2007 Ramez Elmasri, Shamkant B. Navathe

Summary of SQL Syntax

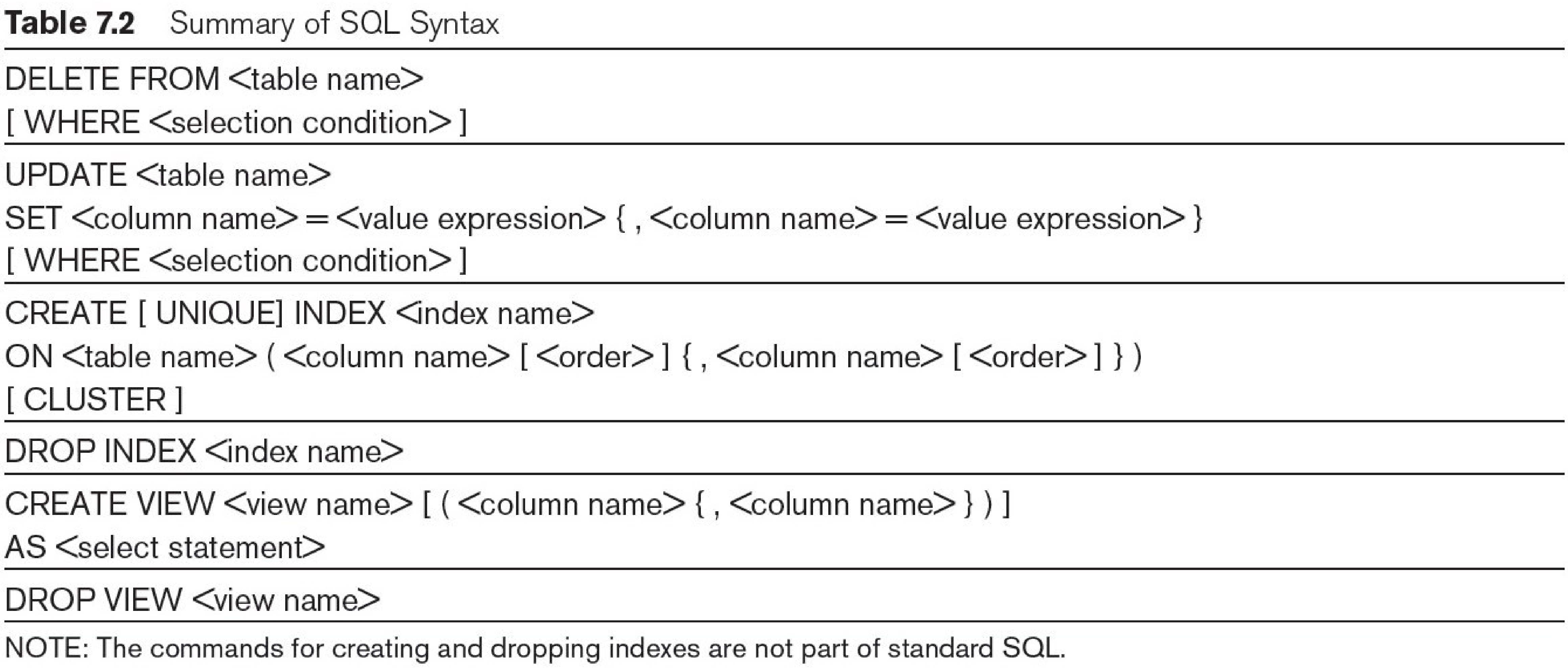


Table 7.2 Summary of SQL Syntax

```
DELETE FROM <table name>
[ WHERE <selection condition> ]

UPDATE <table name>
SET <column name> = <value expression> { , <column name> = <value expression> }
[ WHERE <selection condition> ]

CREATE [ UNIQUE] INDEX <index name>
ON <table name> ( <column name> [ <order> ] { , <column name> [ <order> ] } )
[ CLUSTER ]

DROP INDEX <index name>

CREATE VIEW <view name> [ ( <column name> { , <column name> } ) ]
AS <select statement>

DROP VIEW <view name>
```

NOTE: The commands for creating and dropping indexes are not part of standard SQL.

Copyright @2007 Ramez Elmasri, Shamkant B. Navathe

Summary of SQL Syntax

- Complex SQL:
  - Nested queries, joined tables (in the FROM clause), outer joins, aggregate functions, grouping
- Handling semantic constraints with `CREATE ASSERTION` and `CREATE TRIGGER`
- `CREATE VIEW` statement and materialization strategies
- Schema Modification for the DBAs using `ALTER TABLE , ADD and DROP COLUMN, ALTER CONSTRAINT` etc.

Copyright @2007 Ramez Elmasri, Shamkant B. Navathe

SUBSTRING() Function

- The SUBSTRING() extracts a substring with a specified length starting from a location in an input string.
- The syntax of the SUBSTRING() function:

  SUBSTRING(input_string, start, length);
- In this syntax:
  - input_string can be a character, binary, text, ntext, or image expression.
  - start is an integer that specifies the location where the returned substring starts. **Note that the first character in the input_string is 1, not zero.**
  - length is a positive integer that specifies the number of characters of the substring to be returned. The SUBSTRING() function raises an error if the length is negative. **If start + length > the length of input_string, the substring will begin at the start and include the remaining characters of the input_string.**

SUBSTRING() examples-1

A) Using SUBSTRING() function with literal strings

- This example extracts a substring with the length of 6, starting from the fifth character, in the 'SQL Server SUBSTRING' string.

```
SELECT SUBSTRING('SQL Server SUBSTRING', 5, 6) result;
```

- output:

```
result
------
Server
(1 row affected)
```

SUBSTRING() examples-2

B) Using SUBSTRING() function with table columns

- The example returns the names attribute in the first column, the first letter of the name attribute in the second column, and the third and fourth characters in the final column.

```
SELECT name, SUBSTRING(name, 1, 1) AS Initial ,
SUBSTRING(name, 3, 2) AS ThirdAndFourthCharacters
FROM sys.databases
WHERE database_id < 5;
```

- Output:

name	Initial	ThirdAndFourthCharacters
master	M	St
tempdb	T	Mp
model	M	De
msdb	M	Db

REPLACE() function

- Replace in SQL is a built-in function that allows you to replace all the incidents of a substring within a specified string with a new substring.
- To replace all occurrences of a substring within a string with a new substring.
- Syntax

```
REPLACE(input_string, substring, new_substring);
```

- In this syntax:
    - input_string is any string expression to be searched.
    - substring is the substring to be replaced.
    - new_substring is the replacement string.
- The REPLACE() function returns a new string in which all occurrences of the substring are replaced by the new_substring.
- It returns NULL if any argument is NULL.

Distinct Keyword

- The SQL DISTINCT keyword is used in conjunction with the SELECT statement to eliminate all the duplicate records and fetching only unique records.
- There may be a situation when you have multiple duplicate records in a table, it makes more sense to fetch only those unique records instead of fetching duplicate records.
- Syntax

```
SELECT DISTINCT column1, column2,.....columnN
FROM table_name
WHERE [condition]
```

SQL - LIKE Operator

- The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.
- There are two wildcards often used in conjunction with the LIKE operator:
    - The percent sign (%) represents zero, one, or multiple characters
    - The underscore sign (_) represents one, single character
- **You can also combine any number of conditions using AND or OR operators.**
- Syntax-

```
SELECT column1, column2, ...
FROM table_name
WHERE columnN LIKE pattern;
```

Examples :LIKE operators with '%' and '_' wildcards:

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

Practice Exercise #1.1: (create)

Q1: Create the table suppliers that contains four columns: supplier_id, supplier_name, city, state and supplier_id is primary key

Solution:

```
CREATE TABLE suppliers
( supplier_id int NOT NULL,
  supplier_name char(50) NOT NULL,
  city char(50),
  state char(25),
  CONSTRAINT suppliers_pk PRIMARY KEY (supplier_id)
);
```

Practice Exercise #1.2 (insert)

Q2: Insert some records in the table suppliers that contains four columns: supplier_id, supplier_name, city, state and supplier_id is primary key

Solution:

```
INSERT INTO suppliers   (supplier_id, supplier_name, city, state)
VALUES  (100, 'Microsoft', 'Redmond', 'Washington’),
        (200, 'Google', 'Mountain View', 'California’),
        (300, 'Oracle', 'Redwood City', 'California’),
        (400, 'Kimberly-Clark', 'Irving', 'Texas’),
        (500, 'Tyson Foods', 'Springdale', 'Arkansas’),
        (600, 'SC Johnson', 'Racine', 'Wisconsin’),
        (700, 'Dole Food Company', 'Westlake Village', 'California’),
        (800, 'Flowers Foods', 'Thomasville', 'Georgia’),
        (900, 'Electronic Arts', 'Redwood City', 'California');
```

Practice Exercise #1.3: (select)

Q3: Based on the supplier table, select the unique city values that reside in the state of California and order the results in descending order by city:

- Solution:

```
SELECT DISTINCT city
FROM suppliers
WHERE state = 'California'
ORDER BY city DESC;
```

- These are the results that you should see:

city
Westlake Village
Redwood City
Mountain View

Practice Exercise #1.4: (update)

Q4: Based on the suppliers table populated with the following data, update the city to 'Boise' and the state to "Idaho" for all records whose supplier_name is "Microsoft".

Solution: UPDATE suppliers SET city = 'Boise’,
state = 'Idaho’ WHERE supplier_name = 'Microsoft’;

The supplier table
would now look like thi

supplier_id	supplier_name	city	state
100	Microsoft	**Boise**	**Idaho**
200	Google	Mountain View	California
300	Oracle	Redwood City	California
400	Kimberly-Clark	Irving	Texas
500	Tyson Foods	Springdale	Arkansas
600	SC Johnson	Racine	Wisconsin
700	Dole Food Company	Westlake Village	California
800	Flowers Foods	Thomasville	Georgia
900	Electronic Arts	Redwood City	California

Practice Exercise #1.5: (delete)

Q5: Based on the suppliers table, delete the supplier record whose state is 'California' and supplier_name is not Google:

Solution: DELETE FROM suppliers WHERE state = 'California’
AND supplier_name <> 'Google’;

There would be 3 records deleted and the suppliers table would now look like this:

supplier_id	supplier_name	city	state
100	Microsoft	Redmond	Washington
200	Google	Mountain View	California
400	Kimberly-Clark	Irving	Texas
500	Tyson Foods	Springdale	Arkansas
600	SC Johnson	Racine	Wisconsin
800	Flowers Foods	Thomasville	Georgia

Practice Exercise #2

Sample *employees* table for the queries:

```
CREATE TABLE employees
( employee_number int NOT NULL,
  last_name char(50) NOT NULL,
  first_name char(50) NOT NULL,
  salary int,
  dept_id int, emailid varchar(100),
  CONSTRAINT employees_pk PRIMARY KEY (employee_number)
);
```

Practice Exercise #2:

```
INSERT INTO employees
(employee_number, last_name, first_name, salary, dept_id, emailid)
VALUES
(1001, 'Smith', 'John', 62000, 500, ‘smith@gmail.com’),
(1002, 'Anderson', 'Jane', 57500, 500, ‘anderson@gmail.com’),
(1003, 'Everest', 'Brad', 71000, 501, ‘everest@ny.com’),
(1004, 'Horvath', 'Jack', 42000, 501, ‘horvath@yahoo.com’);
(1005, ‘Mohan', ‘Radha’, 55000, 502, ‘mohan@yahoo.com’);
```

Practice Exercise #2:

1. Select all fields from the *employees* table whose salary is less than or equal to $52,500 (no sorting is required)
2. Select the details of the employee whose first name starts with B.
3. Print name and Id of employee whose email-id is in Gmail.
4. Select the details of the employee who work either for department 501 or 502.
5. Select all the employees working in department 501 in descending order of their salary.

Solution for Practice Exercise #2

- Solution 1:

  SELECT * FROM employees WHERE salary <= 52500;
- Output:

employee_number	last_name	first_name	Salary	dept_id	**Emailid**
1004	Horvath	Jack	42000	501	horvath@yahoo.com

- Solution 2:

  SELECT * FROM employees WHERE first_name like ‘b%’
- Output:-

employee_number	last_name	first_name	salary	dept_id	**emailid**
1003	Everest	Brad	71000	501	Everest@ny.com

Solution for Practice Exercise #2

- Solution 3:

  SELECT employee_number as E_no, last_name, first_name FROM employees where emailid like '%@gmail.com';

- Output

E_no	last_name	first_name
1001	smith	John
1002	Anderson	Jano

- Solution 4:

SELECT * FROM employees WHERE dept_id=501 or dept_id=502; or

SELECT * FROM employees WHERE dept_id in (501, 502)

Output:-

employee_number	last_name	first_name	salary	dept_id	**emailid**
1003	Everest	Brad	71000	501	Everest@ny.com
1004	Horvath	Jack	42000	501	Horvath@yahoo.com
1005	Mohan	Radha	55000	502	mohan@yahoo.com

Solution for Practice Exercise #2

- Solution 5:
  SELECT * FROM employees WHERE dept=501 ORDER BY salary DESC;

- Output

employee_number	last_name	first_name	salary	dept_id	**emailid**
1003	Everest	Brad	71000	501	Everest@ny.com
1004	Horvath	Jack	42000	501	Horvath@yahoo.com

Practice Exercise #3

Sample *customers* table:

Cust_id	First_name	Last_name	Gender	Phone_number
100	Steven	Austin	M	515.123.4567
101	Neena	Singh	F	515.124.4568
102	Bruce	King	M	516.124.4569
103	David	Russell	M	590.123.4560
104	Den	Lee	M	590.124.4561
105	John	Gates	M	590.423.4565
106	Amit	Banda	M	515.124.4550
107	Sundar	Bhatt	M	590.124.4566
108	Clara	Doran	F	515.123.4562

Practice Exercise #3:

1. Write a query to update the portion of the phone_number in the customers table, within the phone number the substring '124' will be replaced by '999’.
2. Write a query to get the details of the customers where the length of the first name greater than or equal to 6.
3. Write a query to extract the last 4 character of phone numbers.
4. Write a query that displays the first name and the length of the first name for all customers whose name starts with the letters ‘D', 'J' or ‘N'. Give each column an appropriate label. Sort the results by the employees' first names.
5. Write a query to display the length of first name for employees where last name contain character ‘a' after 2nd position.

Solution Practice Exercise #3:

- Solution Q1:

```
UPDATE customers SET phone_number = REPLACE(phone_number, '124', '999')
WHERE phone_number LIKE '%124%’;
```

- Solution Q2:

```
SELECT * FROM customers WHERE LENGTH(first_name) >= 6;
```

- Solution Q3:

```
SELECT SUBSTRING(phone_number, 9, 4) as ‘Ph.no’ FROM customers;
```

- Solution Q4:

```
SELECT first_name "Name", LENGTH(first_name) "Length" FROM customers
WHERE first_name LIKE ‘D%' OR first_name LIKE ‘J%' OR first_name LIKE ‘N%'
ORDER BY first_name ;
```

- Solution Q5:

```
SELECT first_name, last_name FROM customers WHERE INSTR(last_name,’A') > 2;
```

Thank you

Copyright @2007 Ramez Elmasri, Shamkant B. Navathe

BITS Pilani
Pilani Campus

DATABASE DESIGN

SESSION - 8

innovate achieve lead

BITS Pilani
Pilani Campus

DEPENDENCY AND NORMALIZATION

Learning Objectives

- What is Functional Dependency?
- Different types of dependencies
  - (i) Functionally dependent
  - (ii) Partially dependent
  - (iii) Transitive dependent
- What is Normalization ?
- Different types of Normal forms
  - i. First normal form
  - ii. Second normal form
  - iii. Third normal form
  - iv. Boyce Codd normal form
- Exercises

Functional Dependency

What is a Functional Dependency?

Functional Dependency (FD) determines the relation of one attribute to another attribute in a database management system (DBMS) system.

Functional dependency helps you to maintain the quality of data in the database. A functional dependency is denoted by an arrow →.

If attribute A functionally determines attribute B we write this as A→B.

Functional Dependency plays a vital role to find the difference between good and bad database design.

Assume that an employee table consists of the data about employees who are working in a company as

Functional Dependency

EMPLOYEE

EmpNo	EmpName	Dept	Salary
E101	Zaheer	HR	78000
E102	Basheer	Production	67000
E103	Naveen	Sales	54000
E110	Siva	System	60000
E112	Shankar	Production	39000

In this example, if we know the value of EmpNo(Employee number), we can obtain EmpName, Dept and salary.

By this, we can say that the EmpName, Dept and Salary are **functionally dependent** on Employee number.

Functional Dependency

It is represented as follows

EmpNo → EmpName, Dept ,Salary

In other words, we can say that for each EmpNo, there is a single value for attributes EmpName, Dept and Salary.

There are various types of functional dependencies:

1. Fully Functional dependency
2. Partial Functional dependency
3. Transitive Functional dependency

The dependencies which are used in normalization process are

1. Partial dependency and
2. Transitive dependency

Full Functional Dependency

In full functional dependency an attribute or a set of attributes uniquely determines another attribute or set of attributes.

If a relation R has attributes X, Y, Z with the dependencies X->Y and X->Z which states that those dependencies are fully functional.

Example:

In the given Employee relation, if we know the value of EmpNo(Employee number), we can obtain EmpName, Dept and salary.

By this, we can say that the EmpName, Dept and Salary are **fully functionally dependent** on Employee number.

Partial Functional Dependency

Partial Dependency

Assume that a table consists of two attributes are combined for forming a primary key.

All non key attributes may either depend on both the key attributes or depend on any one of the key attribute.

If all non key attributes in the table are functionally dependent on any one of the key attributes (not dependent on both), the dependency is called as partial dependency

Example :

Assume that STUDENT table consists of student information as follows:

Partial Functional Dependency

STUDENT **Key attributes** **Non-key attributes**

StuID	**FacultyID**	FacName	Course
S001	F1001	Mr. A	DBSA
S101	F1005	Mr. B	Datastructures
S006	F1001	Mr. A	DBSA
S010	F0024	Mr.C	OOPS
S001	F1111	Mr.D	Web Technology
S101	F0012	Mr. M	Unix
S101	F0024	Mr.C	OOPS

Partial Functional Dependency

In the above table, the non key attribute FacName is functionally dependent on only FacultyID (not on the other key attribute StuID).

So the table has Partial Dependency.

Note :- If a table has a single key attribute, it does not have partial dependency by default.

Transitive dependency

Transitive dependency

We know that a table consists of both key attributes and non key attributes in which the key attributes are used to identify /locate a record /a tuple uniquely.

All the non key attributes are functionally dependent on the key attribute (s).

There are some instances in which a non key attribute may depend on another non key attribute.

If a table has such type of attribute, the table is said to be having transitive dependency.

Transitive dependency

Example: Assume that a STUDENT table consists of the student records as follows:

STUDENT

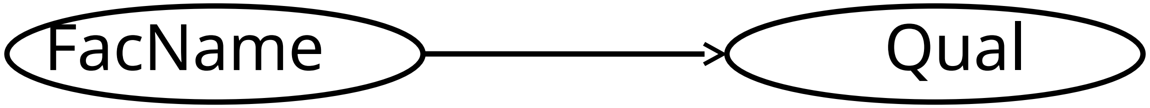
StuID	**FacultyID**	FacName	Qual
S001	F1001	Mr. A	ME , PhD
S101	F1005	Mr. B	ME
S006	F1001	Mr. A	ME , PhD
S010	F0024	Mr.C	Mtech

In the above table, StuID and FacultyID are the key attributes.

FacName and Qual are the non key attributes.

Functional Dependency

The non key attribute Qual is functionally dependent on another non key attribute, FacName.



Hence, the table STUDENT has transitive dependency.

If a table has some dependencies like partial dependency and transitive dependency, there are lot of redundancy (duplication).

Functional Dependency

Advantages of Functional Dependency

The advantages of functional dependency are

(i) Functional Dependency enables to idendify and avoid data redundancy.

Therefore same data do not repeat at multiple locations in the database

(ii) It helps you to maintain the quality of data in the database

(iii) It helps you to defined meanings and constraints of databases

(iv) It helps you to identify bad designs

(v) It helps you to find the facts regarding the database design

Data Normalization

Data normalization is the process of organizing data in a database in order to minimize redundancy and increase data integrity.

Functional dependencies play an important part in data normalization.

With the help of functional dependencies we are able to identify the primary key, candidate key in a table which in turns helps in normalization.

NORMALIZATION

Normalization is the analysis of functional dependency between attributes of user views.

It reduces the complex user view to a set of small and stable subgroups of fields / relations.

This process helps to design a logical data model known as conceptual data model.

There are different levels of normalization as listed below:

1. **Un normalized form (UNF)** – The relation is in *un normalized form* if it satisfies all the properties of a relation
2. **First normal form (1NF)** – The relation is said to be in the *first normal form* if it is already in the un normalized form and has no repeating group.

NORMALIZATION

3. Second normal form (2NF) – The relation is said to be in the *second normal form* if it is already in the first normalized form and has no partial dependency.

In a relation having more than one key field , a sub set of non – key fields may depend on all the key fields but another subset / a particular non – key fields may depend on only one of the key fields (ie., not depend on all the key fields).

Such dependency is called as the partial dependency.

4. **Third normal form (3NF)** – The relation is said to be in the *third normal form* if it is already in the second form and has no transitive dependency.

In a relation, there may be dependency among non – key fields.

Such dependency is called as transitive dependency.

NORMALIZATION

5. **Boyce – Codd form (BCNF)** – The relation is said to be in the *Boyce – codd normal form* if it is already in the third normal form and every determinant is a candidate key.

It is a stricter version of the third normal form.

A determinant is any field (simple or composite field) on which some other field is fully functionally dependent.

NORMALIZATION - Example

CASE PROBLEM

Let us consider the two report requirements namely , INVOICE and REORDER REPORT of XYZ book house as shown in the following figures:

XYZ Book House
Chennai – 600 001

Customer N0 : C10001
Customer Name : WIPRO Systems Date : XX / XX / XX
Customer Address : Chennai - 125

Book# Rs.	**Title**	**Author's Name**	**No.of copies**	**Price (Rs.)**	**Amount**
B10001	Production and Operations Management	Benher10	50	250	
B10010	System Simulation Gordon	20	400	8000	
B10011	Engineering Mathematics	Benher	40	300	12000

Invoice of a book company

NORMALIZATION

The XYZ book house supplies the books to the institutions which frequently buy books from them.

The customers are very keen on the data item namely author.

Normalization of Invoice report.

The invoice report is represented in the relation mentioned above.

The relation is named as INVOICE.

This is in un normalized form.

It is represented as

INVOICE (Cus_no, Cus_name, Cus_Add, Book_No, Title, Author_name, Auth_coun, Unit_price)

NORMALIZATION

The amount in the last column of the invoice for each book can be computed by multiplying the number of copies (Qty) with their respective unit price.

Similarly, the grand total at the bottom of the invoice representing the invoice amount can be calculated by summing the entries in the last column of the invoice.

Similarly, the date at the top right corner of the invoice can be printed using the system clock.

A sample instance of the data of the INVOICE relation is shown in the following table.

This is nothing but the file containing all the data items of the INVOICE report.

NORMALIZATION

Sample Invoice Form

Cus_no	Cus_name	Cus_Add	**Book_No**	Title	Author_name	Auth_coun	Unit_price
C1001	SIST	Chennai-1	B1001	Applied OR	Manmohan	INDIA	355
C1001	SIST	Chennai-1	B1010	OOPS with C++	Schildt	US	450
C1001	SIST	Chennai-1	B2000	Java	Herbert	US	525
C1002	RMK	Chennai-10	B2000	Java	Herbert	US	525

The above relation is in un normalized form.

FIRST NORMAL FORM

First normal form (1NF) – The relation is said to be in the *first normal form* if it is already in the un normalized form and has no repeating group.

In the above relation the fields Cus_no , Cus_name and Cus_Add are forming repaeating group.

NORMALIZATION

This will result in redundancy of data for the first three fields.

It leads to inconsistency of data.

Hence the relation can be divided into two sub relations to remove the repeating group as

CUSTOMER (Cus_no , Cus_name and Cus_Add)

CUSTOMER-BOOK(Cus_no , Book_No , Title , Author_name ,

Auth_coun, Unit_price)

The corresponding relations in their normalized form are as follows:

NORMALIZATION

CUSTOMER relation [Relation 2]

Cus_no	Cus_name	Cus_Add
C1001	SIST	Chennai-1
C1001	SIST	Chennai-1
C1001	SIST	Chennai-1
C1002	RMK	Chennai-10

CUSTOMER-BOOK relation [Relation 3]

Cus_no	**Book_No**	Title	Author_name	Auth_coun	Unit_price
C1001	B1001	OOPS with C++	Manmohan	INDIA	355
C1001	B1010	Applied OR	Schildt	US	450
C1001	B2000	Java	Herbert	US	525
C1002	B2000	Java	Herbert	US	525

Now each relation is in 1NF.

NORMALIZATION

SECOND NORMAL FORM

The relation is said to be in the *second normal form* if it is already in the first normalized form and has no partial dependency.

In relation 2 (CUSTOMER relation), the number of key fields is only one and hence there is no scope for partial dependency.

The absence of partial dependency in a relation takes it to the 2NF without any modification.

But in relation 1 (CUSTOMER-BOOK relation), the field Qty depends on Cus_No as well as Book_No.

But the non key fields (Title , Author_name, Unit_price) depend only on Book_No.

This situation is an example of partial dependency.

NORMALIZATION

Partial dependency will result into insertion update and deletion anomalies.

Insertion Anomaly :- In the relation 3, if we want to insert data of a new book , there must be at least one customer buying it.

It means that a new book can be inserted into the CUSTMOER-BOOK relation only when the first customer buys the book.

Update Anomaly :- In the relation, if we want to change any non key fields , it will result into inconsistency because the same is available in more than one record.

This is an example for updation anomaly.

NORMALIZATION

Deletion Anomaly :- Once the customer order is fully satisfied, the customer's record should be deleted.

If the customer is the only person buying a particular book , the information of the book will not be available when his/her record is deleted.

To overcome this, relation 3 is divided into two relations as relation 4 and relation 5 where both of them are in the 2nd Normal form as

SALES(**Cus_no** , Book_No Qty)

BOOK-AUTHOR(**Book_No** , Title , Author_name , Unit_price)

NORMALIZATION

The corresponding relations are:

SALES relation (Relation 4)

Cus_no	**Book_No**
C1001	B1001
C1001	B1010
C1001	B2000
C1002	B2000

BOOK-AUTHOR relation (Relation 5)

Book_No	Title	Author_name	Auth_coun
B1001	Applied OR	Manmohan	INDIA
B1010	OOPS with C++	Schildt	US
B2000	Java	Herber	US
B2000	Java	Herbert	US

In the relations , there are no partial dependencies.

.

NORMALIZATION

So both the relations are in second normal form.

THIRD NORMAL FORM

The relation is said to be in the *third normal form* if it is already in the second form and has no transitive dependency.

In a relation, there may be dependency among non – key fields.

Such dependency is called as transitive dependency.

In relation 4 (SALES relation), there is only one non - key field and hence there is no scope for any dependency between the non – key fields.

It means that there is no transitive dependency.

Hence the relation 3 can be treated to be in 3NF without any modification. But in relation 5 (BOOK-AUTHOR relation), the field Auth_Coun depends on author’s name.

NORMALIZATION

This situation is an example of the transitive dependency.

Transitive dependency will result into insertion update and deletion anomalies.

Insertion Anomaly :- In case, the book has resident authors who are in the process of developing new books, it will be difficult to include the author's detail in relation 5 .

It means that there should be at least one published book to insert the details of a resident author.

Update Anomaly :- Since the author's details are duplicated , there will be significant redundancy and inconsistency.

Some problem arises when the updation is done.

Deletion Anomaly :- If the only book of a resident author is not reprinted , then the respective author's data is lost.

NORMALIZATION

To overcome these anomalies, relation 5 can be divided into two relations as

BOOK (**Book_No** , Title , Author_name)

AUTHOR(**Author_name** , Author_Coun)

The corresponding relations are:

BOOK relation (Relation 6)

Book_No	Title	Author_name
B1001	Applied OR	Manmohan
B1010	OOPS with C++	Schildt
B2000	Java	Herbert
B2000	Java	Herbert

NORMALIZATION

AUTHOR relation (Relation 7)

Author_name	Auth_coun
Manmohan	INDIA
Schildt US	
Herbert US	
Herbert US	

In the relations , there are no partial dependencies.

So both the relations are in second normal form.

For practical applications , it is sufficient to normalize up to 3NF.

NORMALIZATION

4. Boyce Codd Normal form (BCNF)

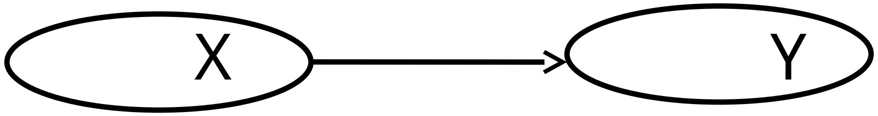
A table is said to be in **Boyce Codd Normal form** iff it is already in third normal form and every determinant in the relation is a candidate key.

Candidate Key – It is an attribute or a collection of attributes that uniquely identify a record

Determinant- The attribute on the left hand size of the arrow in representing the functional dependency is called as the determinant.

Let Y is an attribute that is functionally dependent on another attribute X and is denoted as

NORMALIZATION


X → Y

where the attribute X is the determinant

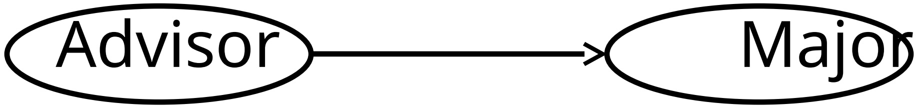
Example: Assume that a student table consists of student data with the following restrictions:

(i) An advisor can handle only one major subject

(ii) A Major Subject may have multiple advisors

According to the restrictions, we identify that Major is functionally dependent on Advisor.

So we have the representation


Advisor → Major

NORMALIZATION

The table is given as

StuID	**AdvisorID**	Major_Sub	Major_Grade
S1	A1	M1	A+
S1	A2	M1	A+
S3	A2	M1	B
S4	A1	M1	C
S5	A2	M1	A

The table is in third normal form because it has no repeating group, no partial dependency and transitive dependency.

NORMALIZATION

But it is not in BCNF because the determinant (Advisor) is not a candidate key (ie. it is only a part of the key) because the combination of the attributes StuID and AdvisorID is used as the candidate key.

To make the table to be in Boyce Codd Normal form , we have to split the table into two tables such as

STUDENT (StudID,AdvisorId, Major_Grade)

Advisor(AdvisorID,Major_Sub)

The records of these tables are

NORMALIZATION

STUDENT

StuID	**AdvisorID**	Major_Grade
S1	A1	A+
S1	A2	A+
S3	A2	B
S4	A1	C
S5	A2	A

ADVISOR

AdvisorID	Major_Sub
A1	M1
A2	M1

In both the tables every determinant is a candidate key. So the

tables are in Boyce Codd Normal form

NORMALIZATION

ADVANTAGES OF NORMALIZATION

Here we can see why normalization is an attractive prospect in RDBMS concepts.

1) A smaller database can be maintained as normalization eliminates the duplicate data. Overall size of the database is reduced as a result.

2) Better performance is ensured which can be linked to the above point. As databases become lesser in size, the passes through the data becomes faster and shorter thereby improving response time and speed.

3) Narrower tables are possible as normalized tables will be fine-tuned and will have lesser columns which allows for more data records per page.

NORMALIZATION

4) Fewer indexes per table ensures faster maintenance tasks (index rebuilds).

5) Also realizes the option of joining only the tables that are needed.

DISADVANTAGES OF NORMALIZATION

1) More tables to join as by spreading out data into more tables, the need to join table’s increases and the task becomes more tedious.

2) Data model becomes extremely difficult to query against as the data model is optimized for applications

Exercise

Let us take an EMPLOYEE table consisting of the employee records as follows:

EMPLOYEE

EmpId	EmpName	DeptId	Salary
E001	Nikhil	D100	76000
E002	Suresh	D106	45200
E003	Kamal	D100	55000
E004	Kavya	D101	49000
E005	Ramya	D108	56240
E006	Keshav	D101	34500
E007	Midhun	D100	49000
E010	Saran	D101	34800

Exercise

Query 1: Write SQL command to display all employee records who are getting salary between Rs.40000 and Rs.60000

Solution:

```
Select *  from EMPLOYEE   where salary >40000 AND
                                        salary <60000;
```

Query 2: Write SQL command to display all employee records

Whose name begins with ‘K’

Solution:

```
Select *  from EMPLOYEE   where  EmpName LIKE ‘K%’;
```

Query 3: Write SQL command to display the number of

departments in the company

Solution:

```
Select distinct( count(DeptId))  from EMPLOYEE;
```

Exercise

Query 4: Write SQL command to add 5% increment in salary those who are getting the salary <= average salary

Solution:

```
Update EMPLOYEE
    set salary=salary+0.05*salary  where
    salary<=(select avg(salary) from EMPLOYEE);
```

Query 5: Write SQL command to display the DeptId that have less than 2 people in it

Solution:

```
Select DeptId, count(DeptId) from  EMPLOYEE where
    group by DeptId having count(DeptId)<2;
```

Exercise

Query 6: Write SQL command to fetch departments along with the total salaries paid for each department

Solution:

```
Select DeptId,sum(salary) from EMPLOYEE group by DeptId;
```

Query 7: Write SQL command to display the name and salary of the employees who earn more than EmpId ‘E004’

Solution:

```
Select EmpName,salary from EMPLOYEE   where
   salary > (select salary from EMPLOYEE where
                    EmpId=‘E004’);
```

Exercise

Query 8: Write SQL command to fetch the list of employees with the same salary

Solution:

```
Select distinct E1.EmpId,E1.EmpName,E1.Salary from
    EMPLOYEE   E1,EMPLOYEE   E2 where
                E1.salary=E2.salary and E1.EmpId!=E2.EmpId;
```

Query 9: Write SQL command to display the employee records whose names are neither ‘Ramya’ nor ‘Midhun’

Solution:

```
Select *  from EMPLOYEE   where
    EmpName not in (‘Ramya’,’Midhun’);
```

Exercise

Assume that a Store has different branches to sell the products.

It does the sales through internet also. The data are given in the following tables:

STORES

Store_location	Sales	**Date**
Chennai	280050	5-5-2020
Bangalore	432000	6-5-2020
Madurai	120000	5-5-2020
Chennai	490000	6-5-2020
Bangalore	343250	10-5-2020
Chennai	123000	15-5-2020

Exercise

INTERNET

Date	Sales
5-5-2020	45000
7-5-2020	34300
8-5-2020	10450
13-5-202	23325
16-5-2020	45000

Examples:

1.If we want to find out all the dates where there is a sales transaction, the SQL command is

```
SELECT Date FROM STORES  UNION SELECT Date FROM      INTERNET;
```

Exercise

2. If we want to find out all the dates where there are both store sales and internet sales, we use the following SQL command

```
SELECT Date FROM STORES INTERSECT
                    SELECT Date FROM INTERNET;
```

3. We want to find out all the dates where there are store sales, but no internet sales. To do so, we use the following SQL statement:

```
SELECT Date FROM STORES MINUS
                  SELECT Date FROM INTERNET;
```

Thanks

BITS Pilani
Pilani Campus

DATABASE DESIGN

SESSION 9

innovate achieve lead

BITS Pilani
Pilani Campus

STORAGE DEVICES AND FILE ORGANIZATION

Learning Objectives

- Different types of storage devices

  (i) Magnetic storage (ii) Optical Storage (iii) Flash memory (iv) Online and Cloud (v) Paper Storage
- What is file organization
- Objectives of file organization
- Types of file organization

  (i) Sequential (ii) Heap (iii) ISAM (iv) B+ tree

  (v) Cluster (vi) Hashing File Organization

STORAGE DEVICES

Devices which are used to store data is known as Storage device.

In computers, the storage devices are classified into

two categories as Primary storage devices and Secondary storage devices.

The devices which are used to store data temporarily, are called primary storage devices.

Example : RAM -Random Access Memory and is a volatile Memory to store temporarily.

The devices which are used to store data permanently , are known as secondary storage devices .

Example: A hard disk -It is a non volatile memory to store data permanently

STORAGE DEVICES

A secondary storage can be removable, internal, or external.

Alternatively referred to as **digital storage**, **storage**, **storage media**, or **storage medium**, a **storage device** is any hardware capable of holding information either temporarily or permanently.

The following picture shows an example of an external Secondary storage device, Drobo.

Drobo
drobo
ComputerHope.com

STORAGE DEVICES

The secondary storage are further classified into the following categories.

1. Magnetic storage
2. Optical Storage
3. Flash memory
4. Online and Cloud
5. Paper Storage

1.Magnetic storage devices - Today, magnetic storage is one of the most common types of storage used with computers. This technology is found mostly on extremely large hybrid hard drives. Examples for magnetic storage devices are : Floppy disks, Hard Drive, Magnetic strip, Tape cassette, etc.,

STORAGE DEVICES

2. **Optical storage devices -** Another common type of storage is optical storage, which uses lasers and lights as its method of reading and writing data.

Examples are: CD-ROM disc, CD-R,CD-RW, Blue-Ray disc, DVD-R, DVD-RW,etc.,

3. **Flash memory devices - Flash memory** has replaced most magnetic and optical media because it is cheaper, more efficient and reliable device.

Examples are : Memory card, Compact Flash, USB Flash drive, Jump drive,SD card, etc.,

4. Online and cloud - Storing data online and in cloud storage is becoming popular as people need to access their data from more than one device.

Examples are Cloud Storage and Network media

STORAGE DEVICES

5. Paper storage - Early computers had no method of using any of the technologies above for storing information and had to rely on paper. Today, these forms of storage are rarely used or found.

Examples are : OMR and Punched Cards

Note : A hard copy could be considered a form of paper storage that is still widely used, although it cannot be easily used to input data back into a computer without the aid of OCR.

FILE ORGANIZATION

File – A file is named collection of related information that is recorded on secondary storage such as magnetic disks, magnetic tables and optical disks.

What is File Organization?

File Organization refers to the logical relationships among various records that constitute the file, particularly with respect to the means of identification and access to any specific record.

In simple terms, Storing the files in certain order is called file Organization.

File Structure refers to the format of the label and data blocks and of any logical control record.

FILE ORGANIZATION

Objectives of File Organization

a. Optimal selection records – records should be accessed as fast as possible
b. Easy Transaction – Any Insert , Update or Delete operation on records in the file should be performed easily without any difficulty
c. Avoid duplication – There should not be any duplicate record
d. Efficient Storing – Records of the file should be stored efficiently in the file so that the cost to perform any operation on the records in the file should be minimum.

FILE ORGANIZATION

Types of File Organizations

There are various methods to organize the files.

Each method has its own advantages and disadvantages on the basis of access or selection .

Thus it is the responsibility of the programmer to decide the best suited file organization method .

Some types of File Organizations are :

1. Sequential File Organization
2. Heap File Organization
3. ISAM method
4. B+ Tree File Organization
5. Clustered File Organization
6. Hash File Organization

FILE ORGANIZATION

We will be discussing each of the file Organizations along with differences and advantages/ disadvantages of each file Organization methods.

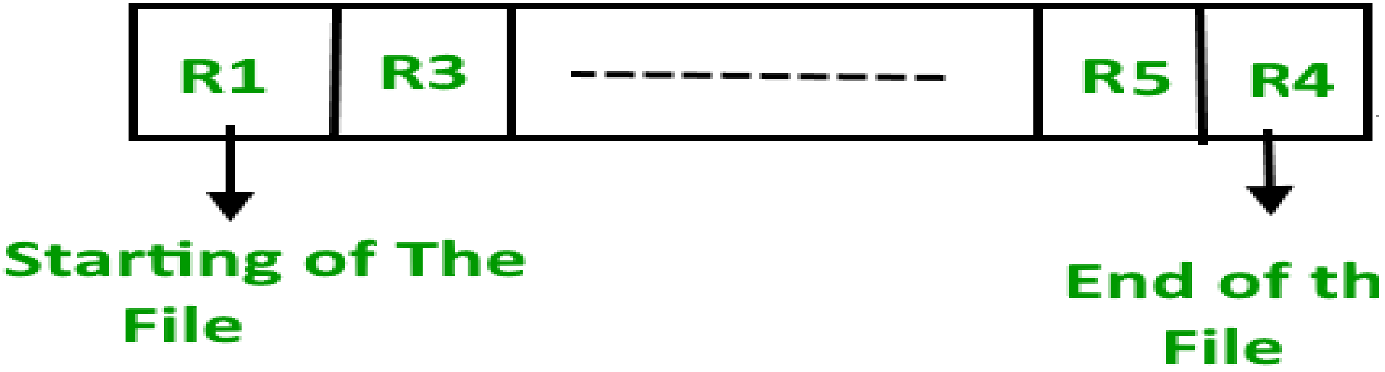
I. Sequential File Organization

The simplest method for file Organization is the Sequential method. In this method the file are stored one after another in a sequential / linear manner.

There are two ways to implement the sequential method:

1. **Pile File Method –** This method is quite simple, in which we store the records in a sequence i.e one after other in the order in which they are inserted into the tables.

FILE ORGANIZATION


R1
R3

R5
R4
Starting of The File
End of th File

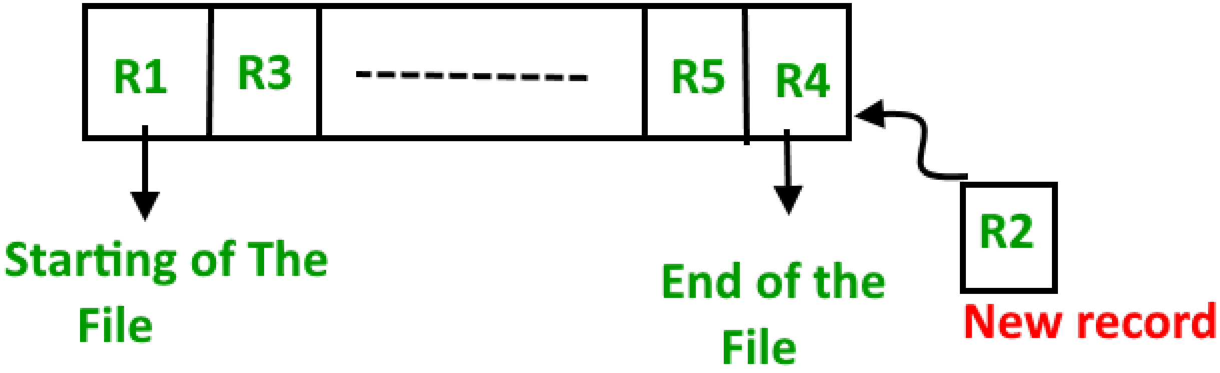
Insertion of new record into the file

Let R1, R3 and so on upto R5 and R4 be records in the sequence.

Here, records are nothing but the tuples or rows in any table.

Suppose a new record R2 has to be inserted in the sequence, then it is simply placed at the end of the file as follows

FILE ORGANIZATION

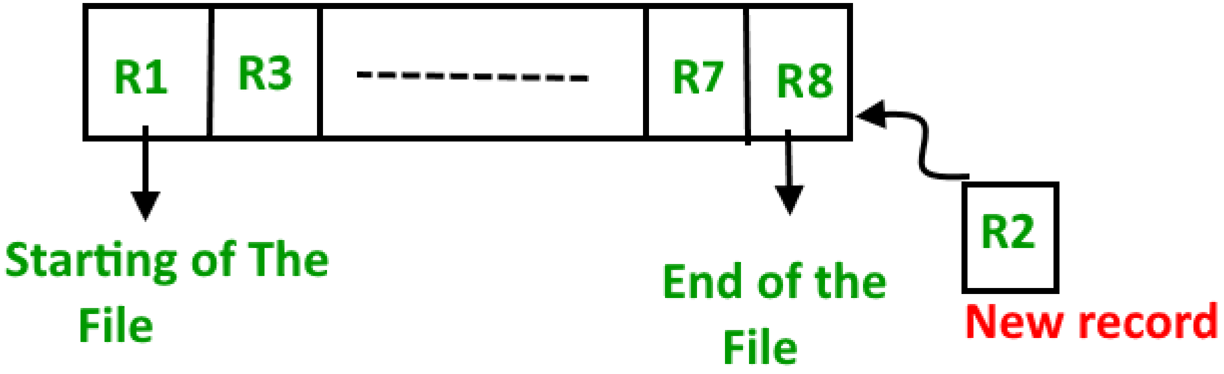

R1
R3

R5
R4
R2
Starting of The File
End of the File
New record

2. Sorted File Method –In this method, As the name itself suggest whenever a new record has to be inserted, it is always inserted in a sorted (ascending or descending) manner.

Sorting of records may be based on any primary key or any other key.

FILE ORGANIZATION


R1
R3

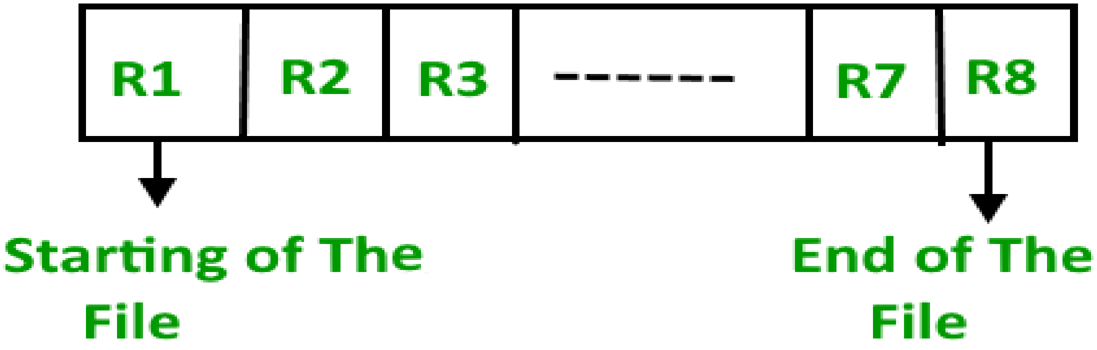
R7
R8
R2
Starting of The File
End of the File
New record

a.Insertion of new record into the file

Let us assume that there is a pre existing sorted sequence of four records R1, R3, and so on up to R7 and R8.

Suppose a new record R2 has to be inserted in the sequence, then it will be inserted at the end of the file and then it will sort the sequence .

FILE ORGANIZATION


R1
R2
R3

R7
R8
Starting of The File
End of The File

Advantages of Sequential File Organization

- Fast and efficient method for huge amount of data.
- It is the simplest design.
- Files can be easily stored in sequential access device like magnetic tapes.
- So these devices are cheaper storage mechanisms.

FILE ORGANIZATION

Disadvantages of Sequential File Organization

- Time is wasted as we cannot go to a particular record randomly but we have to move in a sequential manner. It takes more time
- Sorted file method is inefficient method because it takes much time and space for sorting records(moving the records).

II. Heap File Organization

It is also the simplest form of file organization as sequential file organization.

The records are inserted at the end of the file.

This method does not require any sorting or ordering of the records.

Once the data block is full , the record to be inserted may be stored in any new block.

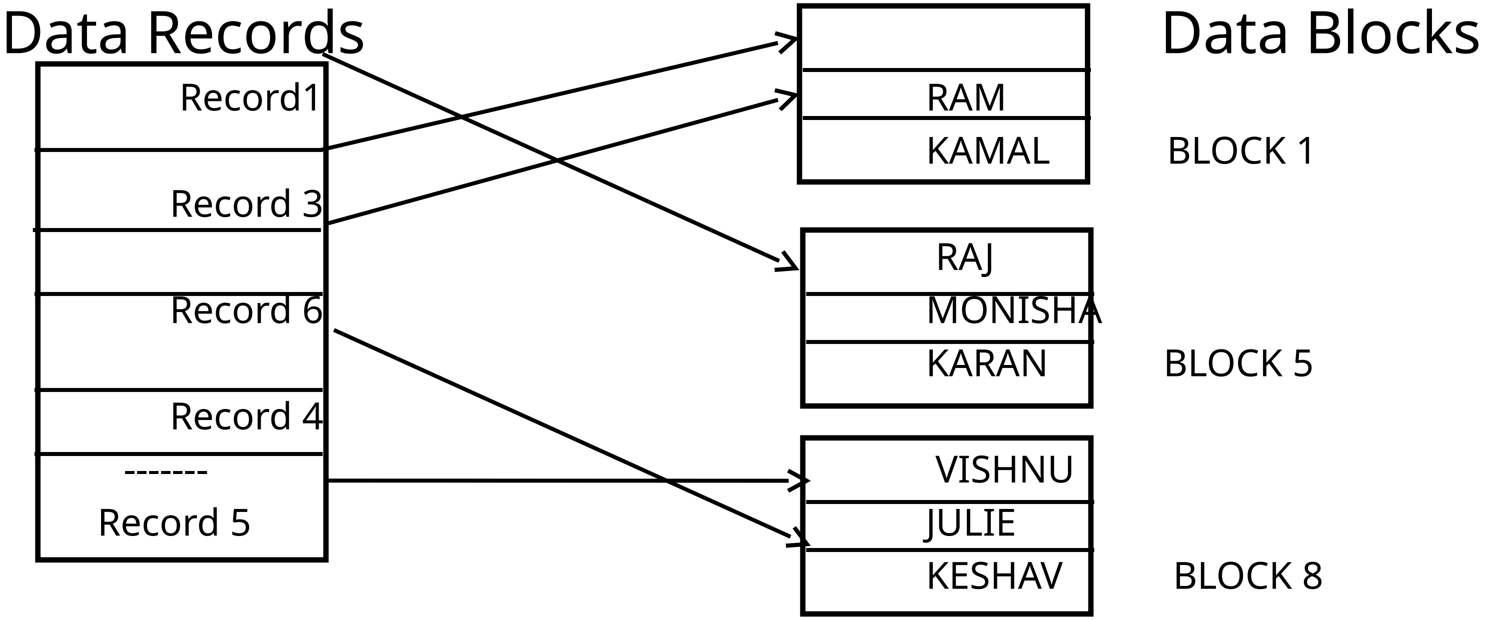
FILE ORGANIZATION

The new block need not be the very next block.

This method is also like the sequential method but data blocks to store are not selected sequentially.

Any block can be selected and it is the responsibility of the DBMS to store and manage the records in the file.

Assume that the records are stored as follows:



Data Records

Record1

Record 3

Record 6

Record 4

Record 5

Data Blocks

RAM

KAMAL

BLOCK 1

RAJ

MONISHA

KARAN

BLOCK 5

VISHNU

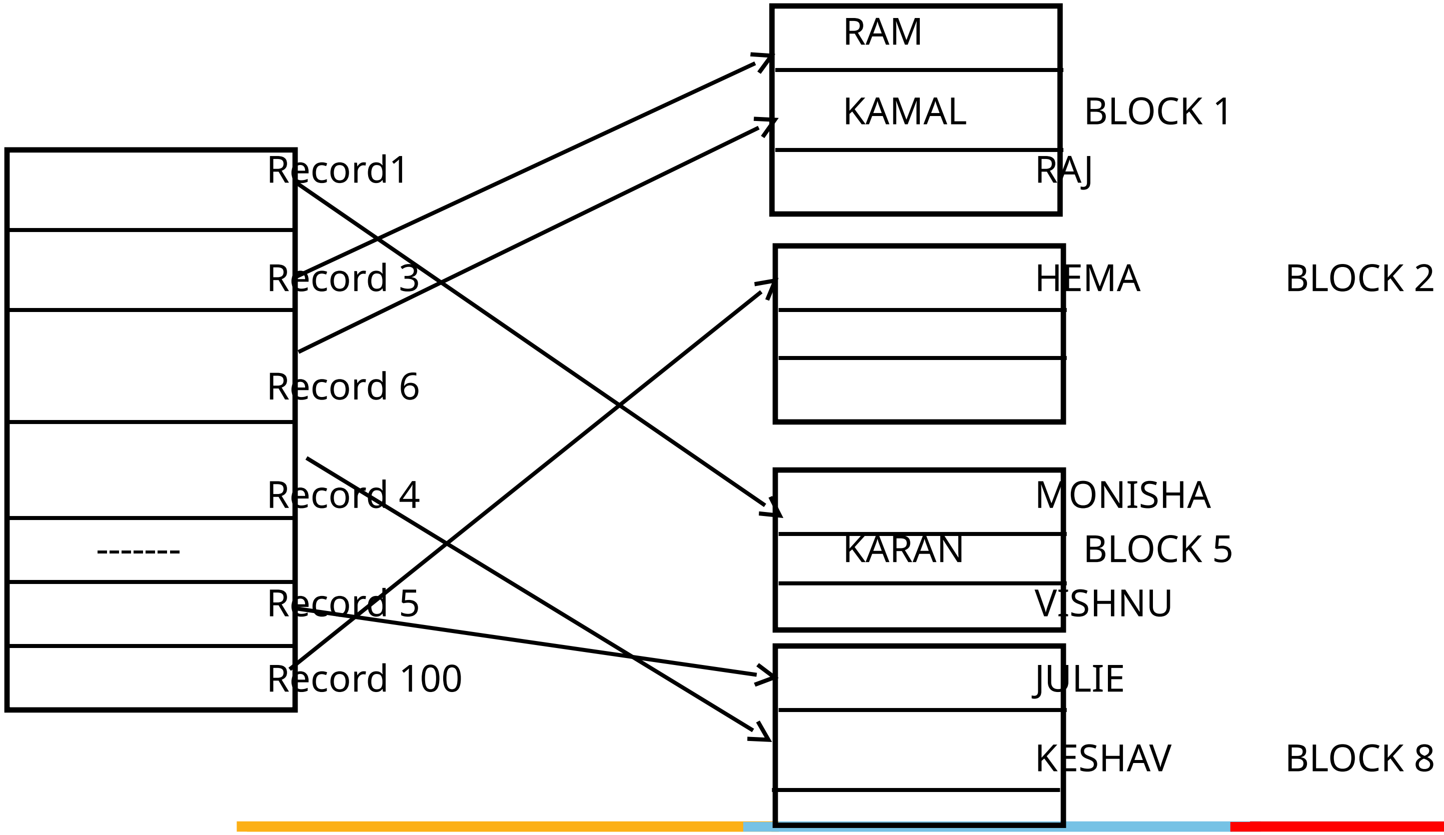
JULIE

KESHAV

BLOCK 8

FILE ORGANIZATION

If a new record is to be inserted , it can be inserted in any of the blocks 2,3,4,6 and 7 or block 8 which has a space to insert the record. When the record (record 100) is inserted , the file is organized in the following format:


Record1
Record 3
Record 6
Record 4

Record 5
Record 100
RAM
KAMAL
RAJ
BLOCK 1
HEMA
BLOCK 2
MONISHA
KARAN
VISHNU
BLOCK 5
JULIE
KESHAV
BLOCK 8

FILE ORGANIZATION

If we want to search, delete or update data in heap file organization then we will traverse the data from the beginning of the file till we get the requested record.

Thus if the database is very huge, searching, deleting or updating the record will take a lot of time.

Advantages of Heap File Organization

- Fetching and retrieving records is faster than sequential record because they can be accessed randomly
- When there is a huge number of data needs to be loaded into the database at a time, then this method of file Organization is best suited.

Disadvantages of Heap File Organization

Problem of unused memory blocks.

Inefficient for larger databases.

FILE ORGANIZATION

III. Indexed Sequential Access Method (ISAM) file organization

ISAM is an advanced sequential file organization method.

The records are stored according to the primary key of the records in the file.

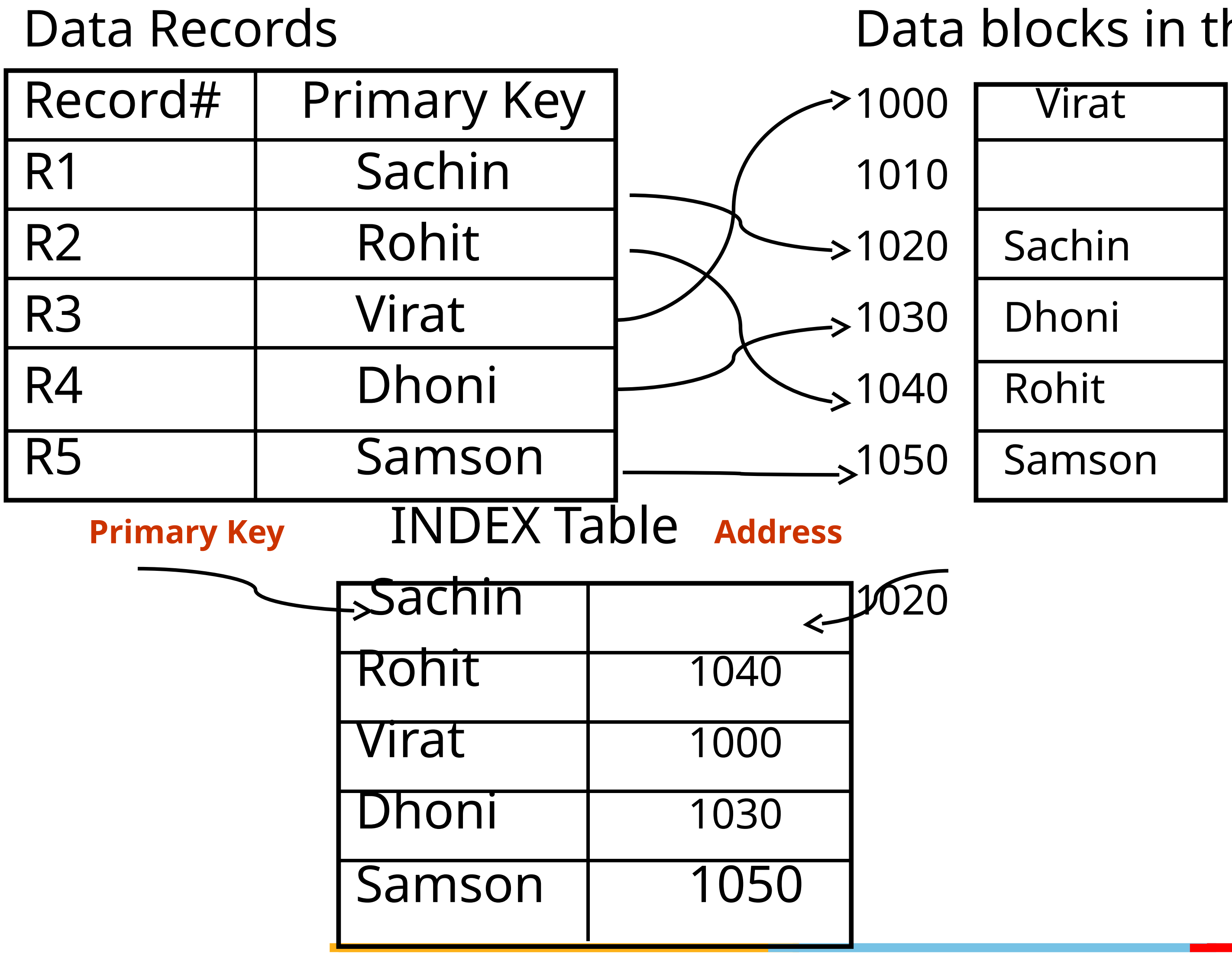
For each primary key , an index value is generated and mapped with the record.

Index is nothing but the address of the record of the file.

Index is a table which has two columns representing Primary key and the address of the record stored in the file.

The records are stored as in the following diagram

FILE ORGANIZATION



Data Records

Record#	Primary Key
R1	Sachin
R2	Rohit
R3	Virat
R4	Dhoni
R5	Samson

Data blocks in the file

1000	Virat
1010	
1020	Sachin
1030	Dhoni
1040	Rohit
1050	Samson

INDEX Table

Primary Key **Address**

Sachin	1020
Rohit	1040
Virat	1000
Dhoni	1030
Samson	1050

FILE ORGANIZATION

Advantages

- It is faster because the records can be accessed randomly since we know their addresses with the help of index table
- It supports range retrieval and partial retrieval. Index is based on the primary key value, we can retrieve data for the given range of value. In the same way, partial value can also be easily searched ie. , Students name starts with ‘Sa’

Disadvantages

- Extra space is required for storing index
- When the records are inserted, the file have to be restructured to maintain the sequence
- When a record is deleted, the space occupied by it should be released. Otherwise it reduces its performance.

FILE ORGANIZATION

IV. B+ Tree File Organization

It is an advanced method of ISAM file organization.

It uses the same concept of key – index combination to store the records but is represented in the tree structure.

B+ tree is similar to binary search tree but it can have more than two leaf nodes.

It stores all the records only at the leaf nodes.

All non terminal nodes (other than leaves) are used to keep the pointer (address) to leaf nodes and are not used for storing the records.

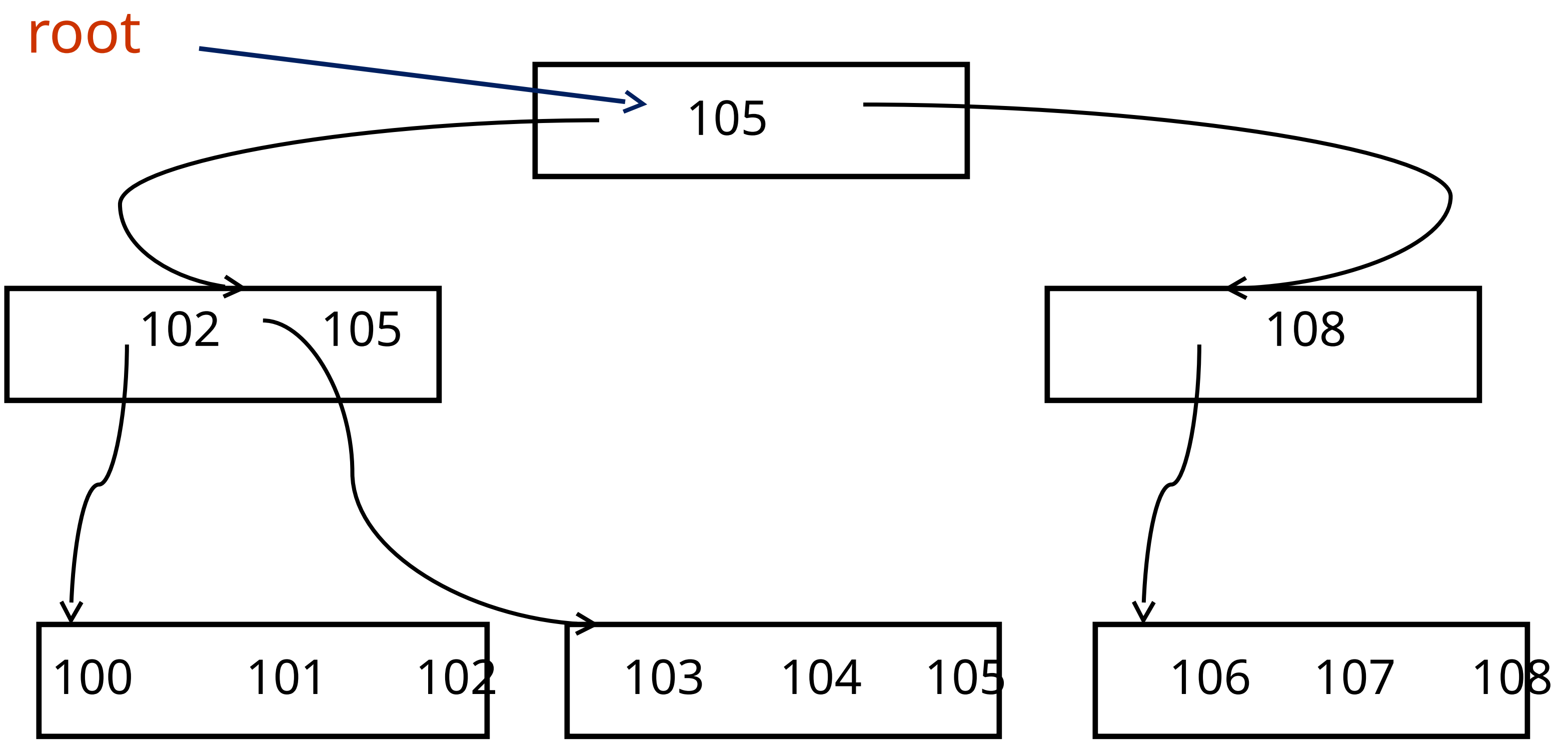
Consider the following table consists of the student records in which StuId is used as the primary key:

FILE ORGANIZATION

StuID	StName	Department
100	Ramesh	Production
101	Hemanth	CSE
102	Jerin	CSE
103	Levin	IT
104	Baskar	CSE
105	Santhosh	IT
106	Meena	Mech
107	David	Production
108	Geetha	CSE

B+ tree is represented as follows:

FILE ORGANIZATION



root

105

102 105

108

100 101 102

103 104 105

106 107 108

Leaves representing the records

FILE ORGANIZATION

- ✓ The above tree has one main node called as the root node. The root node has the key value 105
- ✓ Leaf node contains the data stored in the sorted order
- ✓ Intermediary layers does not have the records actually but they are the pointers to the left child having the data less than or equal to the data in the present node
- ✓ All leaf nodes are balanced. All leaf nodes should exist in the same level

To insert or to delete the records , do the operations on the leaf nodes after searching process finds the correct location to insert.

There should be some adjustments and or insertion on the intermediary nodes.

FILE ORGANIZATION

Advantages of B+ Tree File Organization

- Tree traversal is easier and faster.
- Searching becomes easy as all records are stored only in leaf nodes and are sorted sequential linked list.
- There is no restriction on B+ tree size. It may grows/shrink as the size of data increases/decreases.

Disadvantages of B+ Tree File Organization

- Inefficient for static tables.

FILE ORGANIZATION

V. Cluster File Organization

In cluster file organization, two or more related tables/records are stored within same file known as clusters.

These files will have two or more tables in the same data block and the key attributes which are used to map these table together are stored only once.

Thus it lowers the cost of searching and retrieving various records in different files as they are now combined and kept in a single cluster.

For example we have two tables or relation Employee and Department.

These table are related to each other.

FILE ORGANIZATION

EMPLOYEE

EmpID	EmpName	Address	DeptId
E101	MOHINDER	DELHI	D105
E102	SUNIL	BOMBAY	D102
E103	ROY	CHENNAI	D101
E104	SHYAM	CHENNAI	D103
E105	DRAVID	BANGALORE	D101
E106	SASTRI	BOMBAY	D102
E107	MAMTA	BANGALORE	D104
E108	KAVYA	CHENNAI	D103
E109	RAMYA	TRIVANDRAM	D102

DEPARTMENT

DeptId	DeptName
D101	SALES
D102	PRODUCTION
D103	IT
D104	HR
D105	QM

Therefore these table are allowed to combine using a join operation and can be seen in a cluster file.

FILE ORGANIZATION

CLUSTER KEY

EmpID	EmpName	Address	DeptId	DeptName
E101	MOHINDER	DELHI	D105	QM
E102	SUNIL	BOMBAY	D102	PRODUCTION
E103	ROY	CHENNAI	D101	SALES
E104	SHYAM	CHENNAI	D103	IT
E105	DRAVID	BANGALORE	D101	SALES
E106	SASTRI	BOMBAY	D102	PRODUCTION
E107	MAMTA	BANGALORE	D104	HR
E108	KAVYA	CHENNAI	D103	IT
E109	RAMYA	TRIVANDRAM	D102	PRODUCTION

If we have to insert, update or delete any record we can directly do so. Data is sorted based on the primary key or the key with which searching is done. **Cluster key** is the key with which joining of the table is performed.

FILE ORGANIZATION

VI. Hashing File Organization

It is an efficient technique to directly search the location of desired data on the disk without using index structure.

Data is stored at the data blocks whose address is generated by using hash function.

The memory location where these records are stored is called as data block or data bucket.

Hash File Organization :

Data bucket – Data buckets are the memory locations where the records are stored.

These buckets are also considered as Unit Of Storage.

FILE ORGANIZATION

Hash Function – Hash function is a mapping function that maps all the set of search keys to actual record address.

Generally, hash function uses primary key to generate the hash index – address of the data block.

Hash function can be simple mathematical function to any complex mathematical function.

Hash Index-The prefix of an entire hash value is taken as a

hash index.

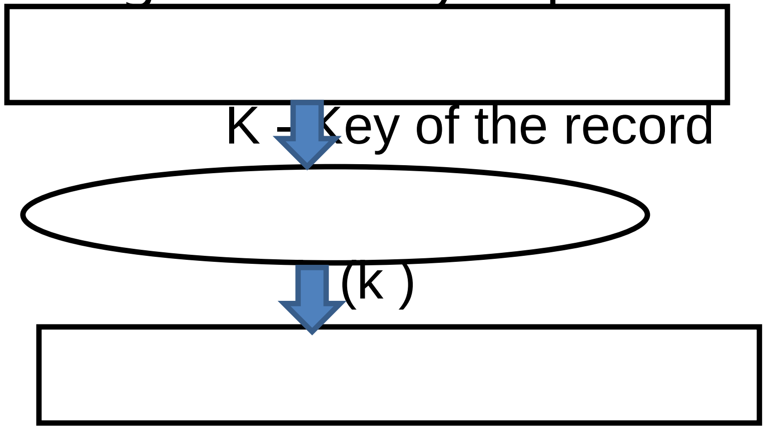
Every hash index has a depth value to signify how many bits are used for computing a hash function.

These bits can address 2n buckets.

When all these bits are consumed, then the depth value is increased linearly and twice the buckets are allocated.

FILE ORGANIZATION

Below given diagram clearly depicts how hash function works:



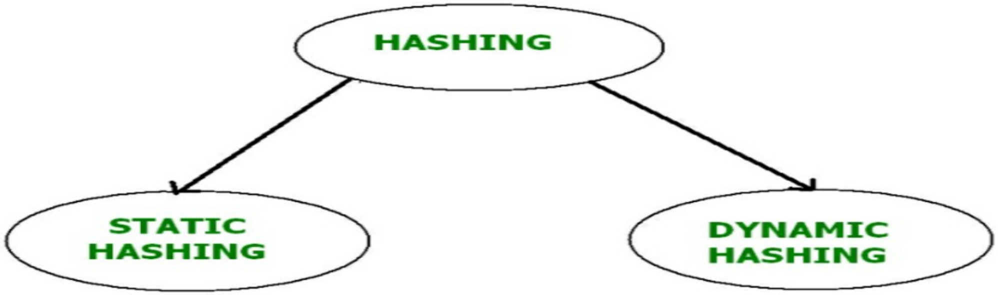
K - Key of the record

(k)

h() is the hashing function

Address to store

Hashing


HASHING
STATIC HASHING
DYNAMIC HASHING

FILE ORGANIZATION

I. STATIC HASHING

In static hashing, when a search-key value is provided, the hash function always computes the same address.

For example, if we want to generate address for STUDENT_ID = 76 using mod (5) hash function, it always result in the same bucket address 1.

There will not be any changes to the bucket address here.

Hence number of data buckets in the memory for this static hashing remains constant throughout.

Operations –

Insertion – When a new record is inserted into the table, The

hash function h generate a bucket address for the new record

based on its hash key K.   Bucket address = h(K)

FILE ORGANIZATION

Searching – When a record needs to be searched, The same hash function is used to retrieve the bucket address for the record.

For Example, if we want to retrieve whole record for ID 76, and if the hash function is mod (5) on that ID, the bucket address generated would be 1. Then we will directly got to address 1 and retrieve the whole record for ID 76.

Here ID acts as a hash key.

Deletion – If we want to delete a record, Using the hash function we will first fetch the record which is supposed to be deleted.

Then we will remove the records for that address in memory.

FILE ORGANIZATION

Updation – The data record that needs to be updated is first searched using hash function, and then the data record is updated.

Now, If we want to insert some new records into the file but if the data bucket address generated by the hash function is not empty or the data already exists in that address.

This becomes a critical situation to handle.

This situation in the static hashing is called **bucket overflow**.

How will we insert data in this case?

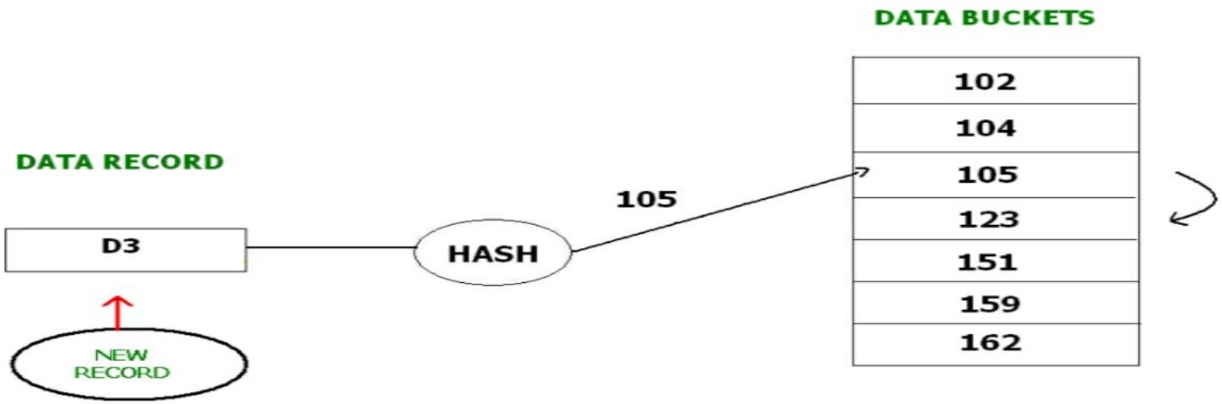
There are several methods provided to overcome this situation.

Some commonly used methods are discussed below:

FILE ORGANIZATION

a. Open Hashing

In Open hashing method, next available data block is used to enter the new record, instead of overwriting older one. This method is also called linear probing. For example, D3 is a new record which needs to be inserted , the hash function generates address as 105. But it is already full. So the system searches next available data bucket, 123 and assigns D3 to it.


DATA BUCKETS
102
104
105
123
151
159
162
DATA RECORD
105
D3
HASH
NEW RECORD

FILE ORGANIZATION

b. Closed hashing

In Closed hashing method, a new data bucket is allocated with same address and is linked after the full data bucket.

This method is also known as overflow chaining.

For example, we have to insert a new record D3 into the tables.

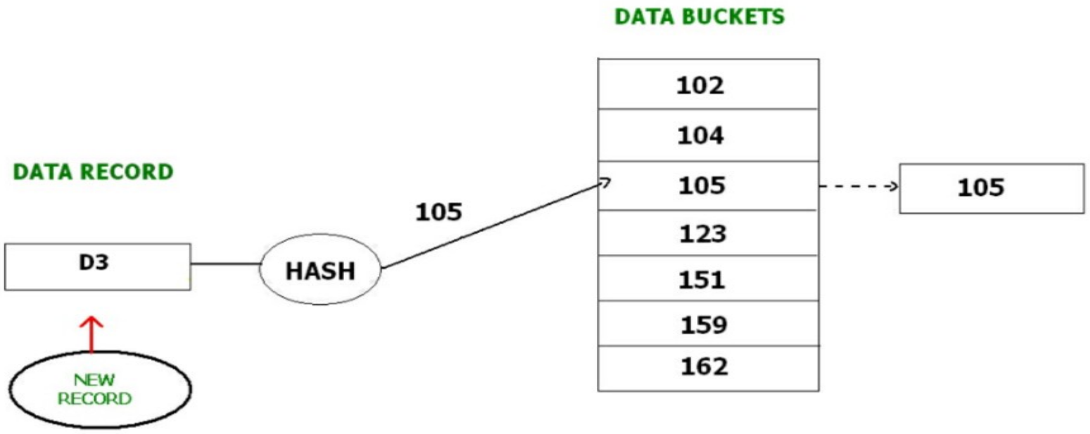
The static hash function generates the data bucket address as 105.

But this bucket is full to store the new data.

In this case, a new data bucket is added at the end of 105 data bucket and is linked to it.

Then new record D3 is inserted into the new bucket.

FILE ORGANIZATION


DATA BUCKETS
102
104
105
123
151
159
162
105
DATA RECORD
105
D3
HASH
NEW RECORD

C. Quadratic probing :

Quadratic probing is very much similar to open hashing or linear probing.

Here, The only difference between old and new bucket is linear.

Quadratic function is used to determine the new bucket address.

FILE ORGANIZATION

d. Double Hashing :

Double Hashing is another method similar to linear probing.

Here the difference is fixed as in linear probing, but this fixed difference is calculated by using another hash function.

That's why the name is double hashing.

II. DYNAMIC HASHING

- The dynamic hashing method is used to overcome the problems of static hashing like bucket overflow.
- In this method, data buckets grow or shrink as the records increases or decreases. This method is also known as Extendable hashing method.
- This method makes hashing dynamic, i.e., it allows insertion or deletion without resulting in poor performance.

FILE ORGANIZATION

Search a key

- First, find the hash address of the key.
- Check how many bits are used in the directory, and these bits are called as i.
- Take the least significant i bits of the hash address.
- This gives an index of the directory.
- Using the index, go to the directory and find bucket address where the record might be.

Insert a new record

- Firstly, you have to follow the same procedure for retrieval, ending up in some bucket.
- If there is still space in that bucket, then place the record in it.

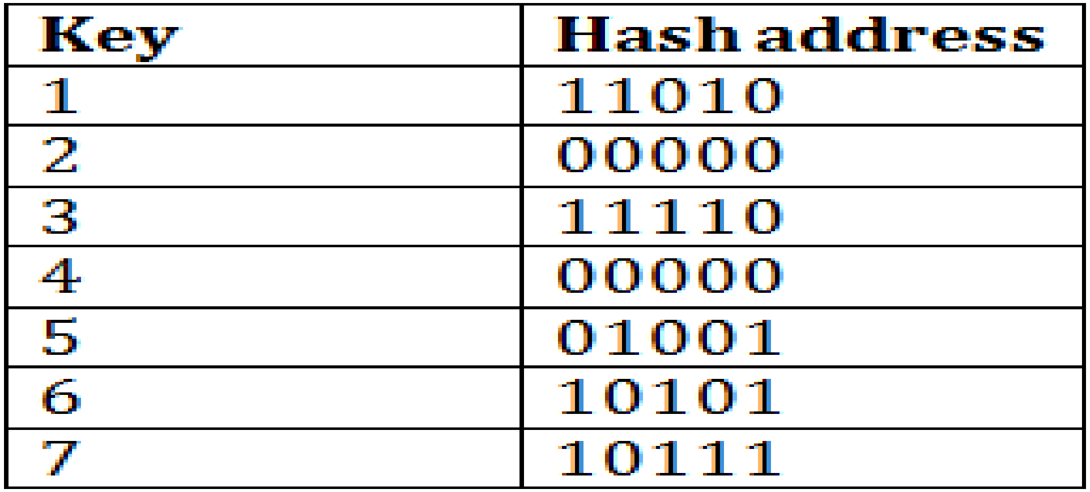
FILE ORGANIZATION

- If the bucket is full, then we will split the bucket and redistribute the records.

For example:

Consider the following grouping of keys into buckets,

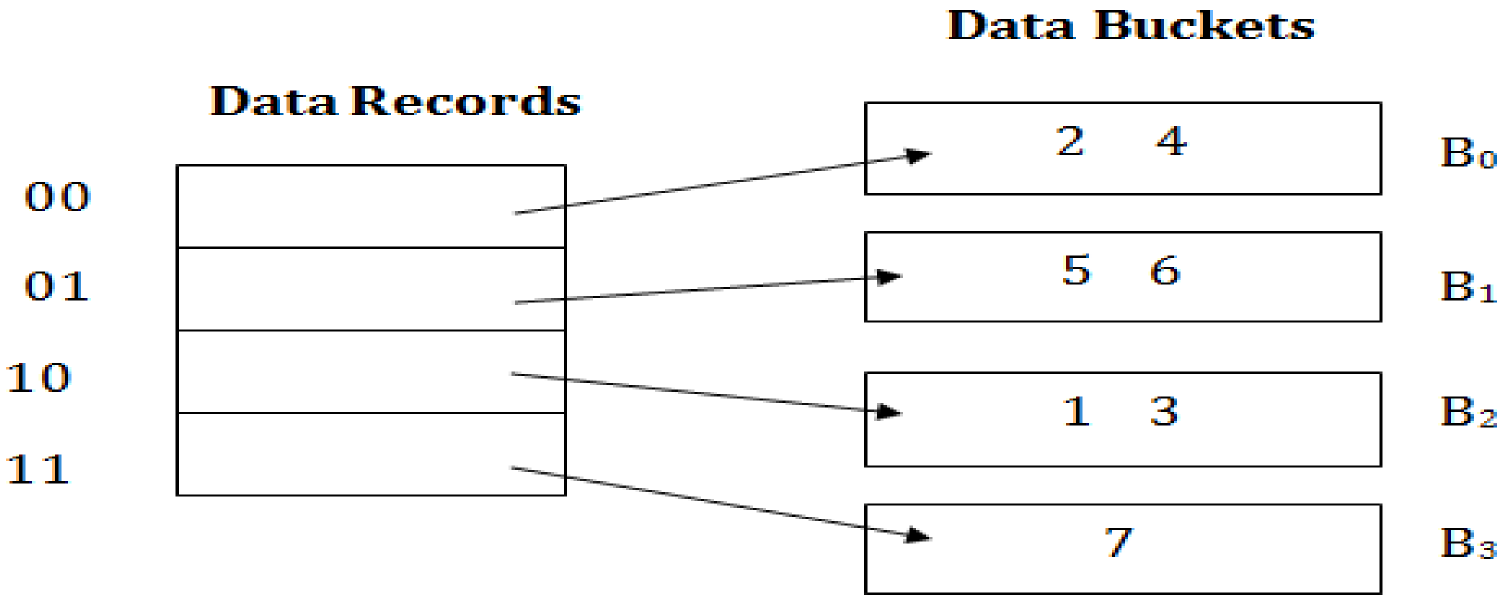
depending on the prefix of their hash address:



Key	Hash address
1	11010
2	00000
3	11110
4	00000
5	01001
6	10101
7	10111

FILE ORGANIZATION

The last two bits of 2 and 4 are 00. So it goes into bucket B0.
The last two bits of 5 and 6 are 01, so it goes into bucket B1.
The last two bits of 1 and 3 are 10, so it goes into bucket B2.
The last two bits of 7 are 11, so it goes into B3.


Data Buckets
Data Records
00
01
10
11
2 4
5 6
1 3
7
B0
B1
B2
B3

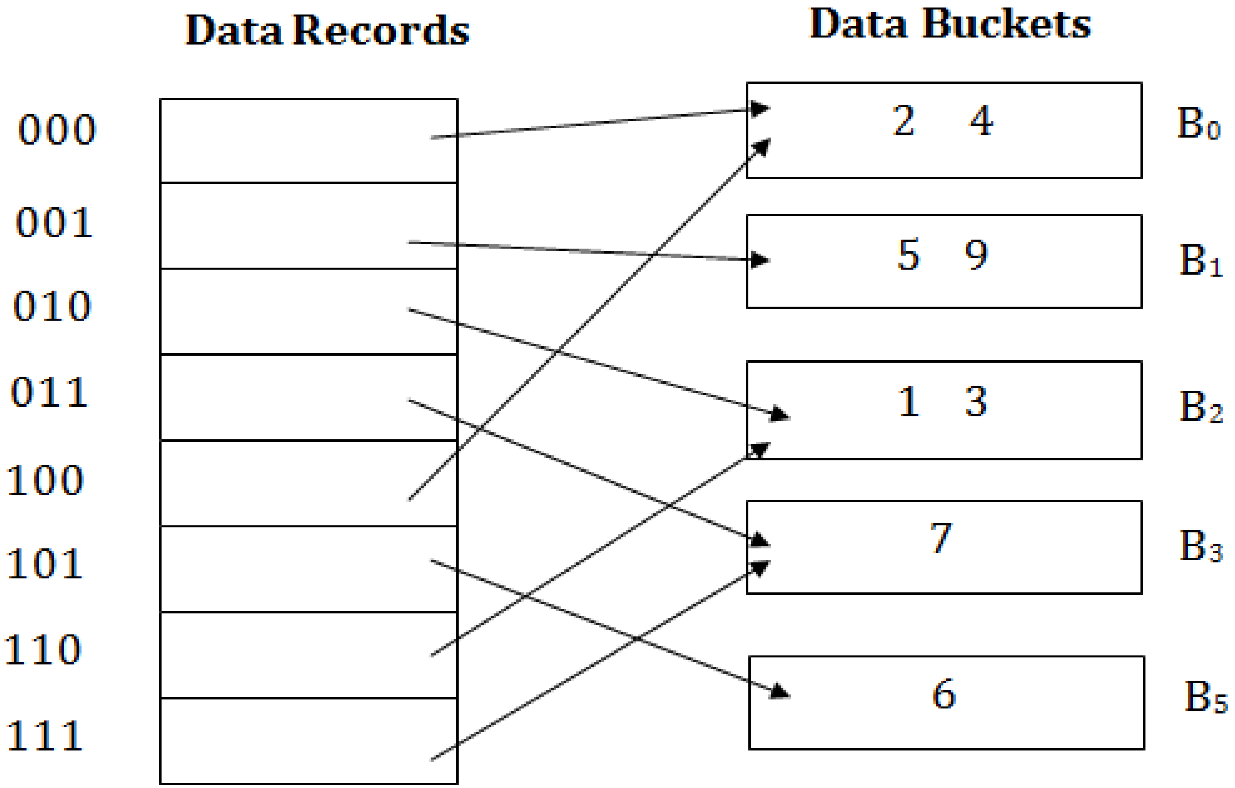
FILE ORGANIZATION

Insert key 9 with hash address 10001 into the above structure:

- Since key 9 has hash address 10001, it has to go into the first bucket. But bucket B1 is full, so it will get split.
- The splitting will separate 5, 9 from 6 since last three bits of 5, 9 are 001, so it will go into bucket B1, and the last three bits of 6 are 101, so it will go into bucket B5.
- Keys 2 and 4 are still in B0. The record in B0 pointed by the 000 and 100 entry because last two bits of both the entry are 00.
- Keys 1 and 3 are still in B2. The record in B2 pointed by the 010 and 110 entry because last two bits of both the entry are 10.

FILE ORGANIZATION

Key 7 are still in B3. The record in B3 pointed by the 111 and 011 entry because last two bits of both the entry are 11.


Data Records
Data Buckets
000
001
010
011
100
101
110
111
2 4
5 9
1 3
7
6
B0
B1
B2
B3
B5

FILE ORGANIZATION

Advantages of dynamic hashing

- In this method, the performance does not decrease as the data grows in the system.
- In this method, memory is well utilized as it grows and shrinks with the data. There will not be any unused memory lying.
- This method is good for the dynamic database where data grows and shrinks frequently.

Disadvantages of dynamic hashing

- In this method, if the data size increases then the bucket size is also increased. These addresses of data will be maintained in the bucket address table. This is because the data address will keep changing as buckets grow and shrink. If there is a huge increase in data, maintaining the bucket address table becomes tedious.
- In this case, the bucket overflow situation will also occur. But it might take little time to reach this situation than static hashing.

Thanks

Database Design

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Contact Session 10: Indexing used in Databases & B+ trees

Recap

- Storage Types and Characteristics
- Secondary Storage Devices
- Placing File Records on Disk
- Hashing Techniques
- Other Primary File Organizations

Reference : Chapter 16 - Disk Storage, Basic File Structures, Hashing, and Modern Storage Architectures (Ramez Elmasri & Shamkant B. Navathe, Fundamentals of Database Systems, Pearson Education, 7th Edition, 2017)

Agenda

- **Buffering of Blocks**
- **Indexing used in Databases with examples.**
- **Understanding B+ trees and**
- **Constructing B+ trees with examples**
- **Construction of B+ trees and Problem Solving**

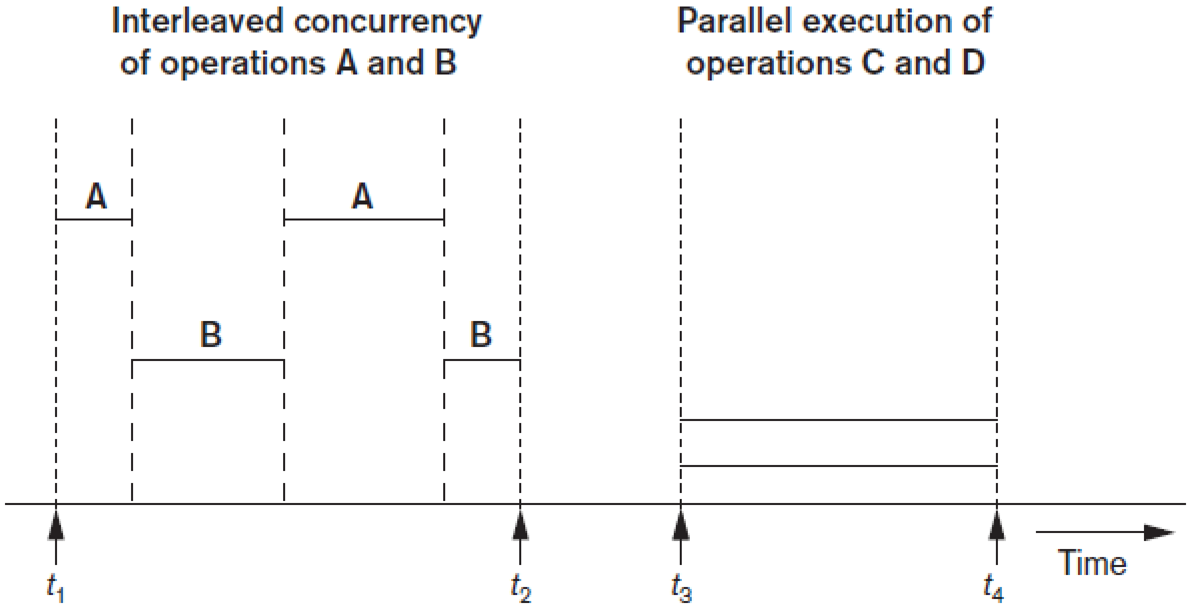
Reference : Chapter 17 - Indexing Structures for Files and Physical Database Design (Ramez Elmasri & Shamkant B. Navathe, Fundamentals of Database Systems, Pearson Education, 7th Edition, 2017)

Buffering of Blocks

A buffer is a memory location used by a database management system (DBMS) to temporarily hold data that has recently been accessed or updated in the database.

This buffer, often referred to as a database buffer, acts as a link between the programs accessing the data and the physical storage devices.

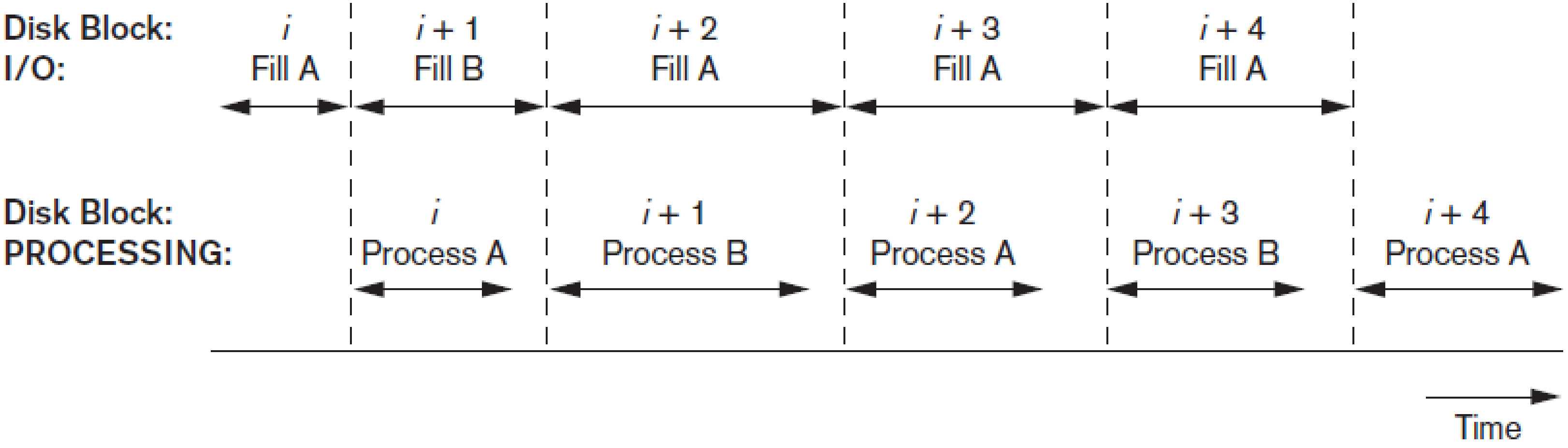
Buffering is most useful when processes can run concurrently in parallel


Interleaved concurrency of operations A and B
Parallel execution of operations C and D
A
A
B
B
t_1
t_2
t_3
t_4
Time

Interleaved concurrency versus parallel execution

Buffering of Blocks (cont’d.)

Double buffering can be used to read continuous stream of blocks


Disk Block:
I/O:
i
Fill A
i + 1
Fill B
i + 2
Fill A
i + 3
Fill A
i + 4
Fill A
Disk Block:
PROCESSING:
i
Process A
i + 1
Process B
i + 2
Process A
i + 3
Process B
i + 4
Process A
Time

Use of two buffers, A and B, for reading from disk

Buffer Management and Replacement Strategies

- Buffer management information
  - Pin count
  - Dirty bit
- Buffer replacement strategies
  - Least recently used (LRU)
  - Clock policy
  - First-in-first-out (FIFO)

Why do we need Buffer replacement strategies?

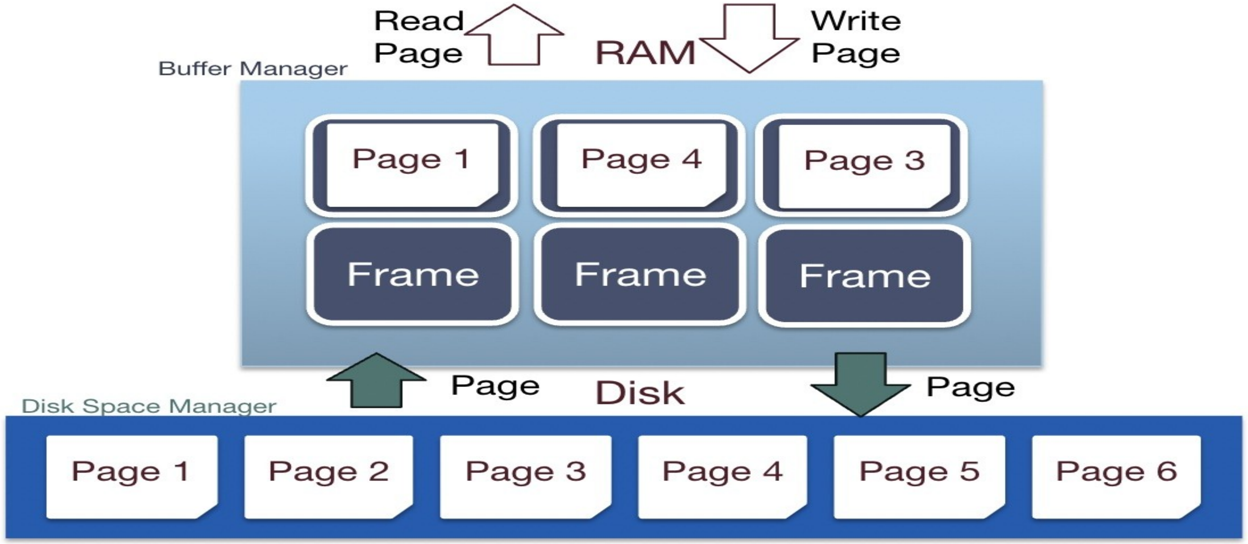
Buffer Management and Replacement Strategies

The buffer manager is responsible for managing pages in memory and processing page requests from the file and index manager.

Since space on memory is limited, so we cannot afford to store all pages in the buffer pool.

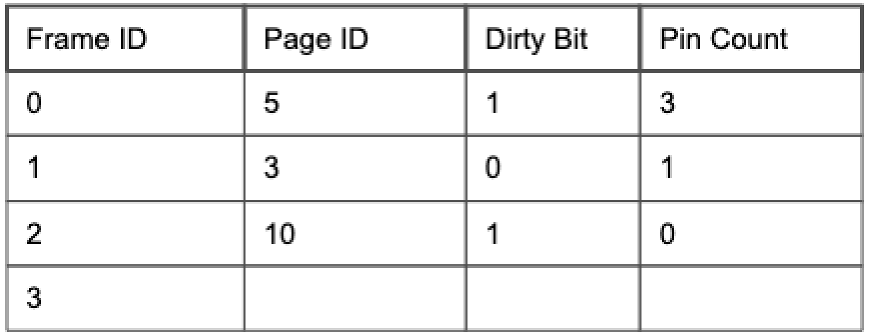
The buffer manager is responsible for the eviction policy, or choosing which pages to evict when space is filled up.

When pages are evicted from memory or new pages are read in to memory, the buffer manager communicates with the disk space manager to perform the required disk operations.


Read Page
RAM
Write Page
Buffer Manager
Page 1
Page 4
Page 3
Frame
Frame
Frame
Page
Disk
Page
Disk Space Manager
Page 1
Page 2
Page 3
Page 4
Page 5
Page 6

Buffer Management and Replacement Strategies

Buffer Pool : Memory is converted into a buffer pool by partitioning the space into frames that pages can be placed in. A buffer frame can hold the same amount of data as a page can (so a page fits perfectly into a frame). To efficiently track frames, the buffer manager allocates additional space in memory for a metadata table.



Frame ID	Page ID	Dirty Bit	Pin Count
0	5	1	3
1	3	0	1
2	10	1	0
3			

The table tracks 4 pieces of information:

1. **Frame ID** that is uniquely associated with a memory address
2. **Page ID** for determining which page a frame currently contains
3. **Dirty Bit** for verifying whether or not a page has been modified
4. **Pin Count** for tracking the number of requestors currently using a page

Placing File Records on Disk (cont’d.)

The records of data that we store are of variable length – Why?

This may be due to the following reasons

Reasons for variable-length records

- One or more fields have variable length
- One or more fields are repeating
- One or more fields are optional
- File contains records of different types

Record Blocking and Spanned Versus Unspanned Records

- File records allocated to disk blocks
- Two strategies can be used for allocation
- Spanned records
  - Larger than a single block
  - Pointer at end of first block points to block containing remainder of record
- Unspanned
  - Records not allowed to cross block boundaries

Record Blocking and Spanned Versus Unspanned Records (cont'd.)

- Blocking factor
  - Average number of records per block for the file

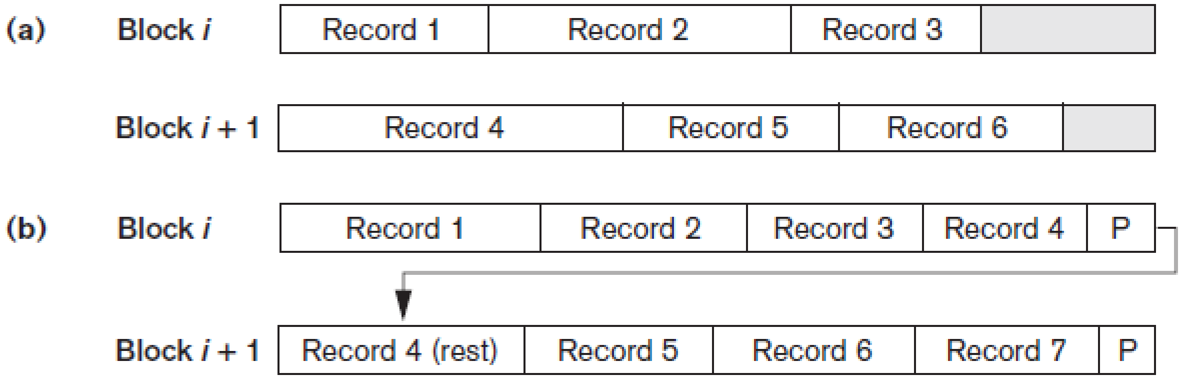

(a)
Block *i*
Record 1
Record 2
Record 3
Block *i* + 1
Record 4
Record 5
Record 6
(b)
Block *i*
Record 1
Record 2
Record 3
Record 4
P
Block *i* + 1
Record 4 (rest)
Record 5
Record 6
Record 7
P

Figure 16.6 Types of record organization (a) Unspanned (b) Spanned

Record Blocking and Spanned Versus Unspanned Records (cont'd.)

- Allocating file blocks on disk
  - Contiguous allocation
  - Linked allocation
  - Indexed allocation

  File header or descriptor is used to allocate and locate files in the memory
- File header (file descriptor)
  - Contains file information needed by system programs
    - Disk addresses
    - Format descriptions

Operations on Files

- Retrieval operations
  - No change to file data
- Update operations
  - File change by insertion, deletion, or modification
- Records selected based on selection condition

Indexing and Indices

- Index / Indices are used to speed up record retrieval in response to certain search conditions
- Index structures provide secondary access paths
- Any field can be used to create an index
  - Multiple indexes can be constructed
- Most indexes based on ordered files
  - Tree data structures organize the index

17.1 Types of Single-Level Ordered Indexes

- Ordered index similar to index in a textbook
- Indexing field (attribute)
  - Index stores each value of the index field with list of pointers to all disk blocks that contain records with that field value
- Values in index are ordered
- Primary index
  - Specified on the ordering key field of ordered file of records

Types of Single-Level Ordered Indexes

- **Clustering index**
  - Used if numerous records can have the same value for the ordering field
- **Secondary index**
  - Can be specified on any non-ordering field
  - Data file can have several secondary indexes

Primary Indexes

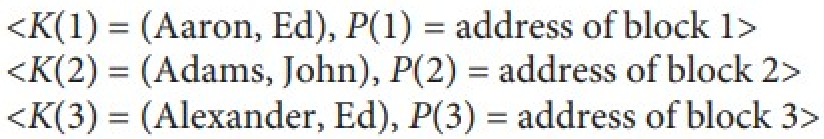
- **Ordered file with two fields**
  - Primary key, *K(i)*
  - Pointer to a disk block, *P(i)*
- **One index entry in the index file for each block in the data file**
- **Indexes may be dense or sparse**
  - Dense index has an index entry for every search key value in the data file
  - Sparse index has entries for only some search values

Primary Indexes

Example: To create a primary index on the ordered file say for a set of students or employees, use the Name field as primary key, because that is the ordering key field of the file (assuming that each value of Name is unique).

Each entry in the index has a Name value and a pointer.

The first three index entries are as follows:



<$K(1)$ = (Aaron, Ed), $P(1)$ = address of block 1>
<$K(2)$ = (Adams, John), $P(2)$ = address of block 2>
<$K(3)$ = (Alexander, Ed), $P(3)$ = address of block 3>

Primary Indexes (cont'd.)

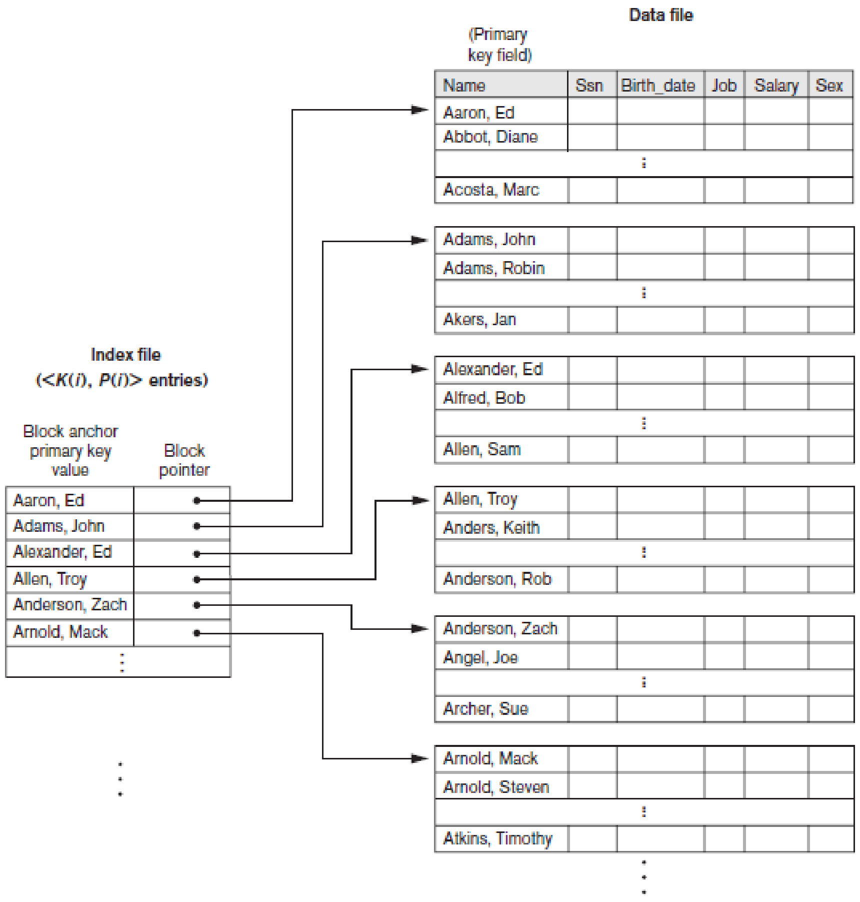

Data file
(Primary key field)
Name
Ssn
Birth_date
Job
Salary
Sex
Aaron, Ed
Abbot, Diane
Acosta, Marc
Adams, John
Adams, Robin
Akers, Jan
Alexander, Ed
Alfred, Bob
Allen, Sam
Allen, Troy
Anders, Keith
Anderson, Rob
Anderson, Zach
Angel, Joe
Archer, Sue
Arnold, Mack
Arnold, Steven
Atkins, Timothy
Index file
(<K(i), P(i)> entries)
Block anchor primary key value
Block pointer
Aaron, Ed
Adams, John
Alexander, Ed
Allen, Troy
Anderson, Zach
Arnold, Mack

Figure 17.1 Primary index on the ordering key field of the file shown in previous slide

Example 1.

Suppose that we have an ordered file with r = 3,00,000 records stored on a disk with block size B = 4,096 bytes.

File records are of fixed size and are unspanned, with record length R = 100 bytes.

The blocking factor for the file would be bfr = $\lfloor (B/R) \rfloor$ = $\lfloor (4{,}096/100) \rfloor$ = 40 records per block.

The number of blocks needed for the file is b = $\lceil (r/bfr) \rceil$ = $\lceil (300{,}000/40) \rceil$ = 7,500 blocks.

A binary search on the data file would need approximately

$$\lceil \log_2 b \rceil = \lceil (\log_2 7{,}500) \rceil = 12.87 \cong 13 \text{ block accesses.}$$

Contd…

Now suppose that the ordering key field of the file is V = 9 bytes long, a block pointer is P = 6 bytes long, and we have constructed a primary index for the file.

The size of each index entry is Ri = (9 + 6) = 15 bytes, so the blocking factor for the index is bfri = $\lfloor(B/Ri)\rfloor = \lfloor(4{,}096/15)\rfloor$ = 273 entries per block.

The total number of index entries ri is equal to the number of blocks in the data file, which is 7,500.

The number of index blocks is hence bi = $\lceil(ri/bfri)\rceil = \lceil(7{,}500/273)\rceil$ = 28 blocks.

To perform a binary search on the index file would need $\lceil(\log_2 bi)\rceil = \lceil(\log_2 28)\rceil = 5$ block accesses.

To search for a record using the index, we need one additional block access to the data file for a total of 5 + 1 = 6 block accesses—an improvement over binary search on the data file, which required 13 disk block accesses.

Primary Indexes (cont’d.)

- Major problem: insertion and deletion of records
  - Move records around and change index values
  - Solutions
    - Use unordered overflow file (≡ Heap)
    - Use linked list of overflow records

Clustering Indexes

- **Clustering field**
  - File records are physically ordered on a non-key field without a distinct value for each record
- **Ordered file with two fields**
  - Same type as clustering field
  - Disk block pointer

Clustering Indexes (cont'd.)

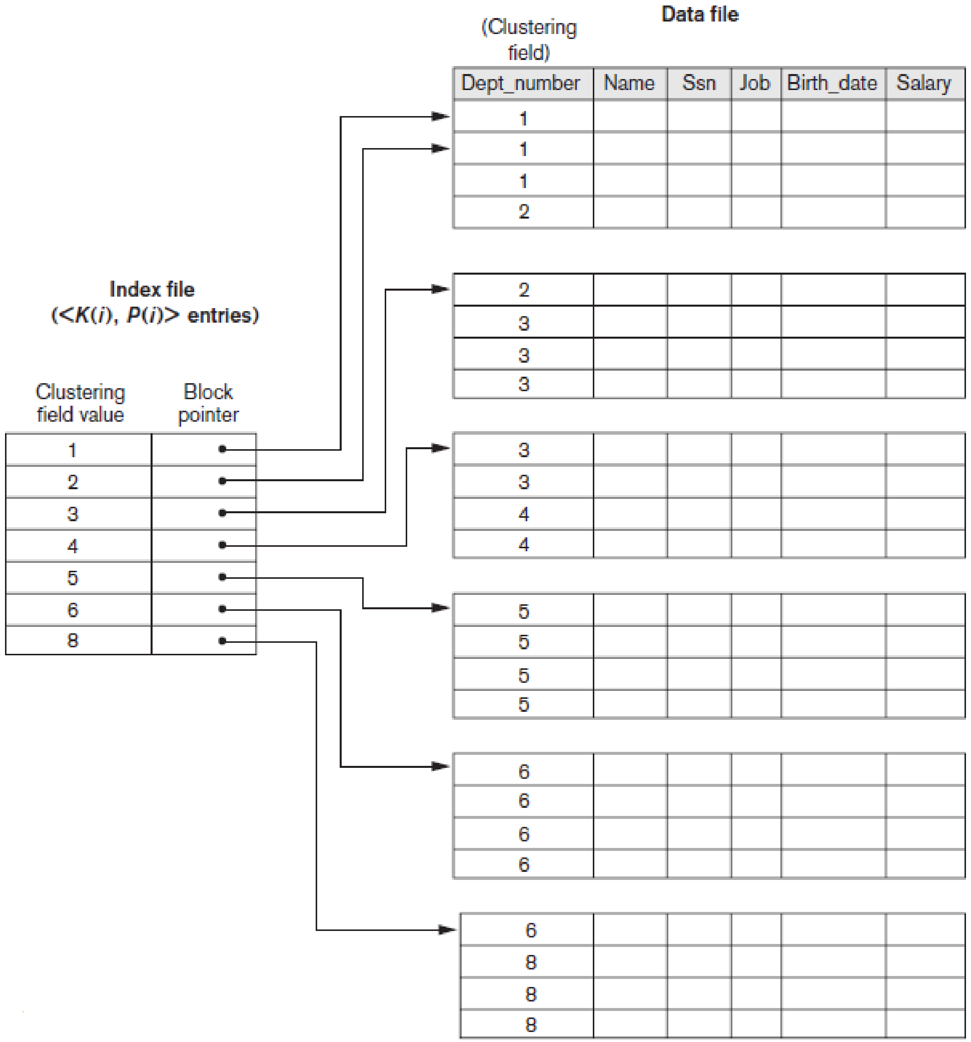

Data file
(Clustering field)
Dept_number
Name
Ssn
Job
Birth_date
Salary
1
1
1
2
2
3
3
3
3
3
4
4
5
5
5
5
6
6
6
6
6
8
8
8
Index file
(<K(i), P(i)> entries)
Clustering field value
Block pointer
1
2
3
4
5
6
8

Figure 17.2 A clustering index on the Dept_number ordering nonkey field of an EMPLOYEE file

Example 2.

Suppose that we consider the same ordered file with r = 300,000 records stored on a disk with block size B = 4,096 bytes.

Imagine that it is ordered by the attribute Zipcode and there are 1,000 zip codes in the file (with an average 300 records per zip code, assuming even distribution across zip codes.)

The index in this case has 1,000 index entries of 11 bytes each (5-byte Zipcode and 6-byte block pointer) with a blocking factor bfri = $\lfloor(B/Ri)\rfloor = \lfloor(4{,}096/11)\rfloor$ = 372 index entries per block.

The number of index blocks is hence bi = $\lceil(ri/bfri)\rceil = \lceil(1{,}000/372)\rceil$ = 3 blocks.

To perform a binary search on the index file would need $\lceil(\log_2 bi)\rceil = \lceil(\log_2 3)\rceil$ = 2 block accesses.

Again, this index would typically be loaded in main memory (occupies 11,000 or 11 Kbytes) and takes negligible time to search in memory.

One block access to the data file would lead to the first record with a given zip code.

Secondary Indexes

- **Provide secondary means of accessing a data file**
  - Some primary access exists
- **Ordered file with two fields**
  - Indexing field, *K(i)*
  - Block pointer or record pointer, *P(i)*
- **Usually need more storage space and longer search time than primary index**
  - Improved search time for arbitrary record

Secondary Indexes (cont'd.)

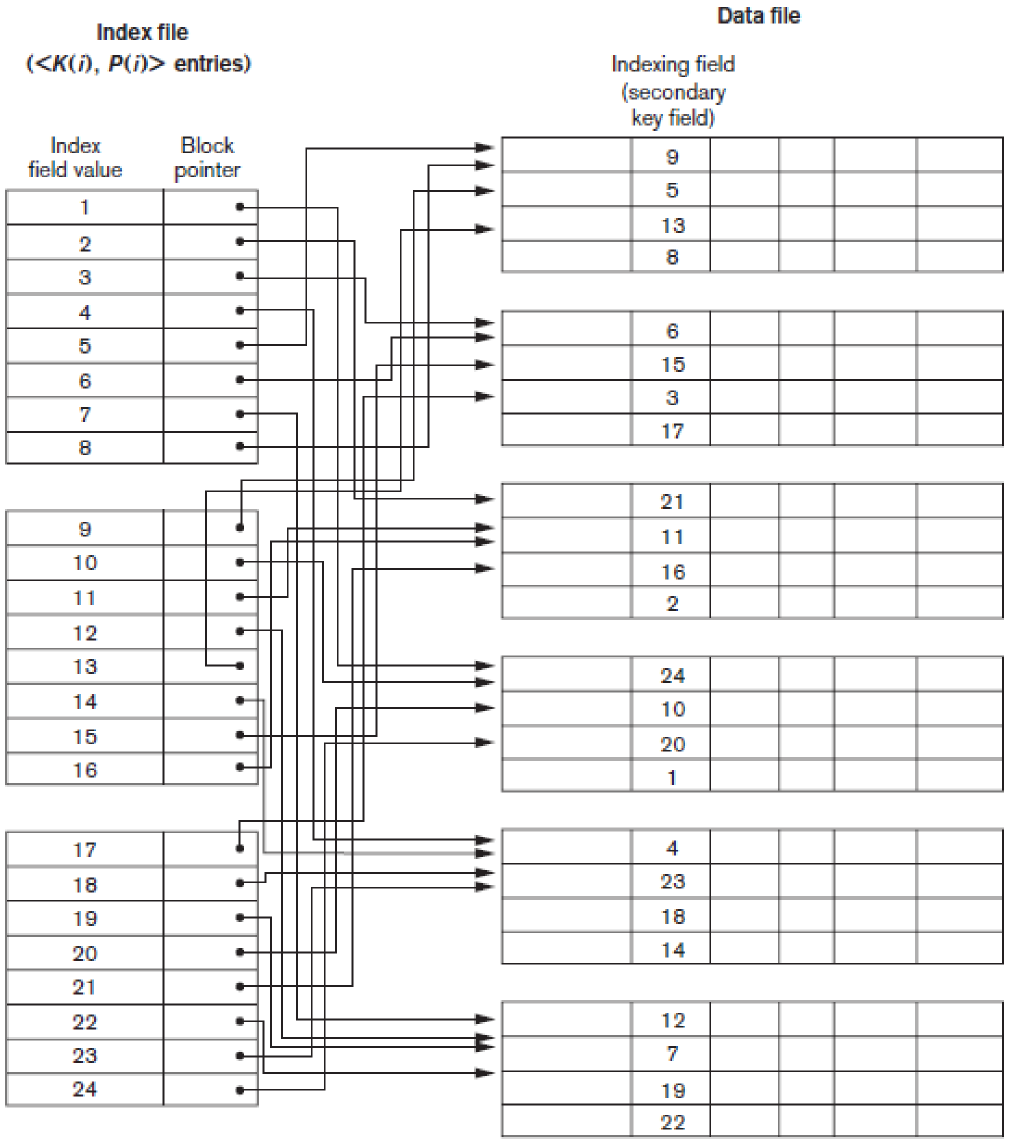
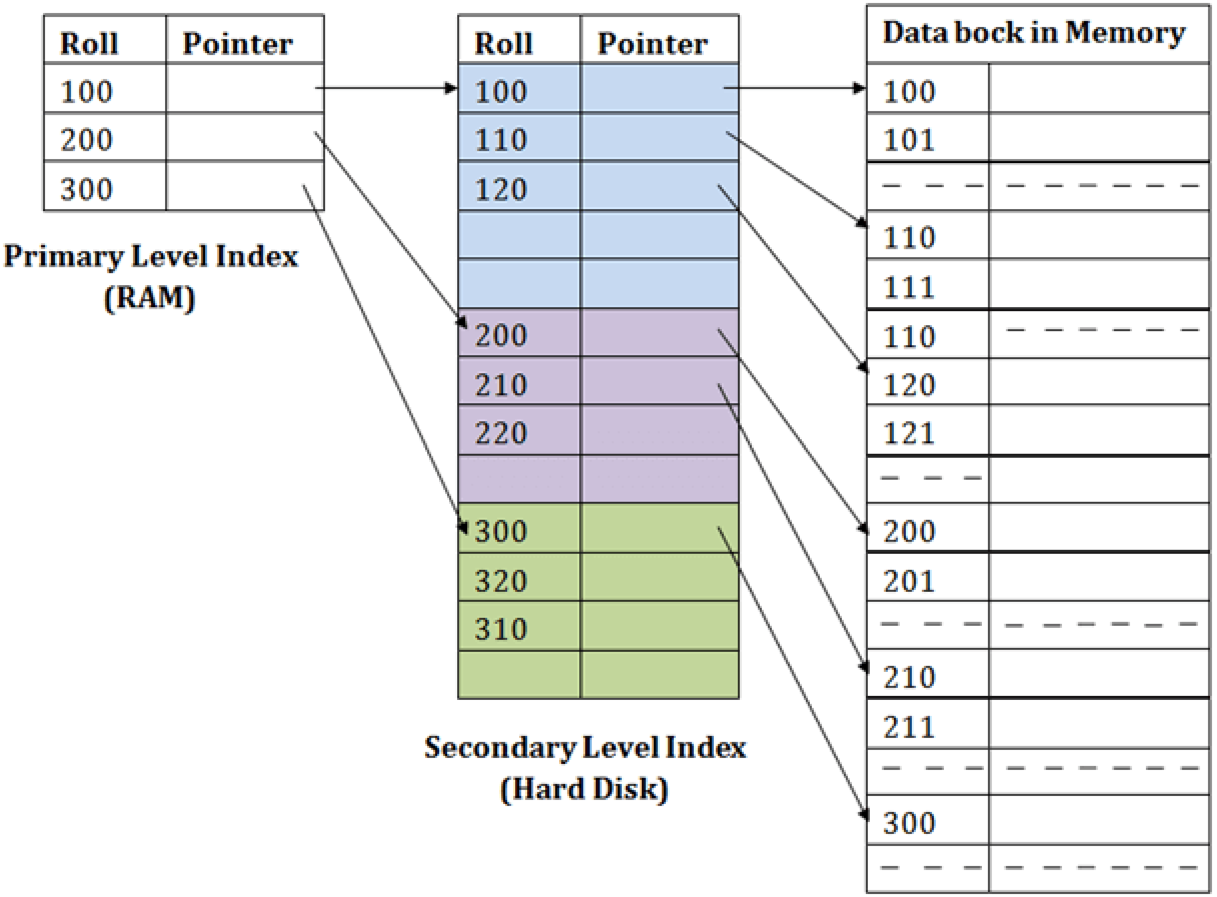

Index file
(<K(i), P(i)> entries)
Index field value
Block pointer
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
Data file
Indexing field (secondary key field)
9
5
13
8
6
15
3
17
21
11
16
2
24
10
20
1
4
23
18
14
12
7
19
22

Figure 17.4 Dense secondary index (with block pointers) on a nonordering key field of a file.

Secondary Indexes (cont’d.)


Roll
Pointer
100
200
300
Primary Level Index
(RAM)
Roll
Pointer
100
110
120
200
210
220
300
320
310
Secondary Level Index
(Hard Disk)
Data bock in Memory
100
101
110
111
110
120
121
200
201
210
211
300

Example 3.

Consider the file of Example 1 with r = 300,000 fixed-length records of size R = 100 bytes stored on a disk with block size B = 4,096 bytes.

The file has b = 7,500 blocks, as calculated in Example 1.

Suppose we want to search for a record with a specific value for the secondary key—a non-ordering key field of the file that is **V = 9 bytes long**.

Without the secondary index, to do a linear search on the file would require b/2 = 7,500/2 = 3,750 block accesses on the average.

Suppose that we construct a secondary index on that non-ordering key field of the file.

As in Example 1, a block pointer is P = 6 bytes long, so each index entry is Ri = (9 + 6) = 15 bytes, and the blocking factor for the index is bfri = ⌊(B/Ri)⌋ = ⌊(4,096/15)⌋ = 273 index entries per block.

In a dense secondary index such as this, the total number of index entries ri is equal to the number of records in the data file, which is 300,000.

The number of blocks needed for the index is hence bi = ⌈(ri/bfri)⌉ = ⌈(300,000/273)⌉ = 1,099 blocks. (As compared to 13 block accesses on the Primary index on the ordering key field and 2 block accesses using clustering index)

17.2 Multilevel Indexes

- **Designed to greatly reduce remaining search space as search is conducted**
- **Index file**
  - Considered first (or base level) of a multilevel index
- **Second level**
  - Primary index to the first level
- **Third level**
  - Primary index to the second level

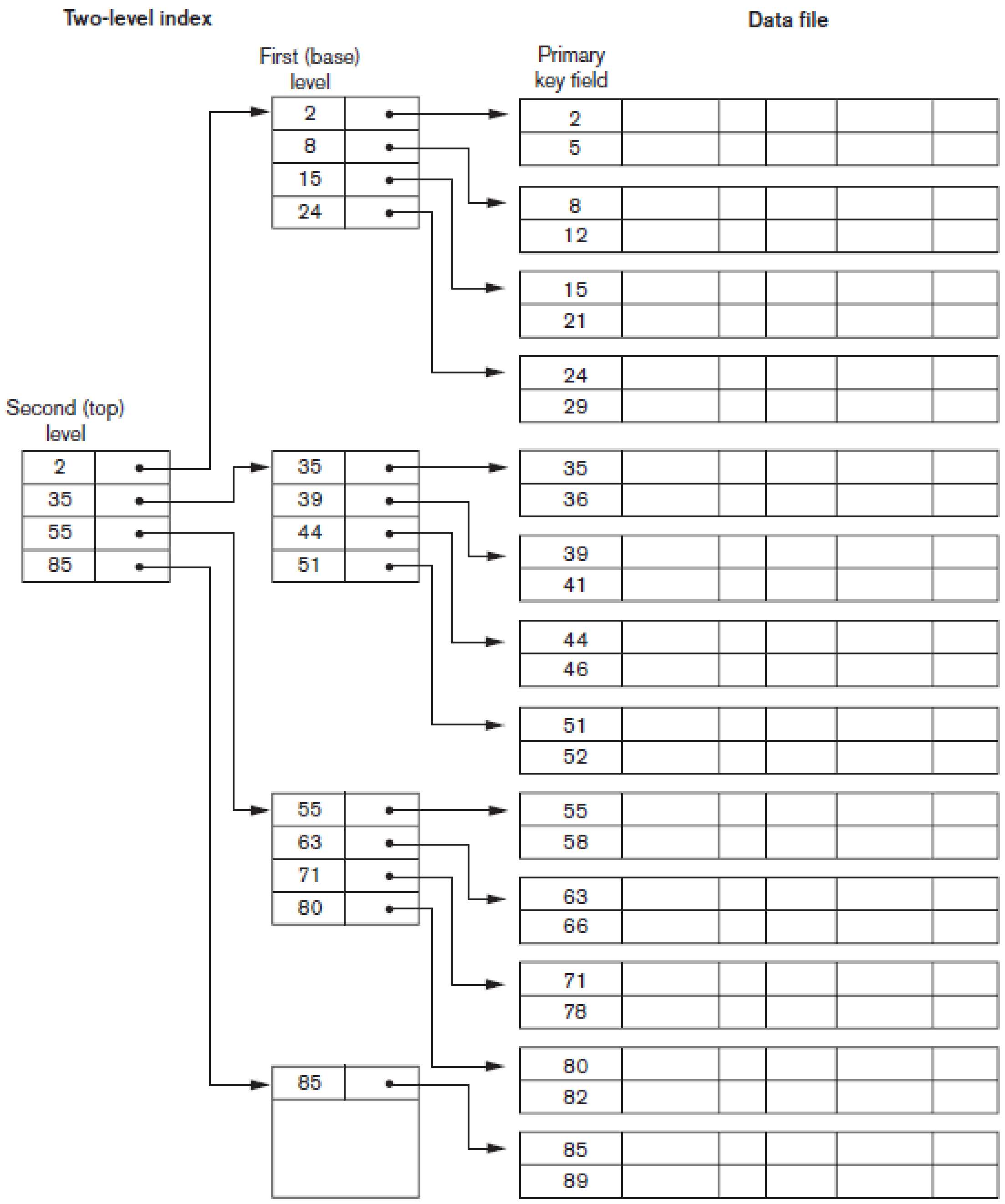

Two-level index
Data file
First (base) level
Primary key field
Second (top) level
2
35
55
85
2
8
15
24
35
39
44
51
55
63
71
80
85
2
5
8
12
15
21
24
29
35
36
39
41
44
46
51
52
55
58
63
66
71
78
80
82
85
89

Figure 17.6 A two-level primary index resembling ISAM (indexed sequential access method) organization

Example 4.

Suppose that the dense secondary index of Example 3 is converted into a multilevel index.

We calculated the index blocking factor bfri = 273 index entries per block, which is also the fan-out for the multilevel index; the number of first-level blocks b1 = 1,099 blocks was also calculated.

The number of second-level blocks will be b2 = ⌈(b1/fo)⌉ = ⌈(1,099/273)⌉ = 5 blocks, and the number of third level blocks will be b3 = ⌈(b2/fo)⌉ = ⌈(5/273)⌉ = 1 block.

Hence, the third level is the top level of the index, and t = 3.

To access a record by searching the multilevel index, we must access one block at each level plus one block from the data file, so we need t + 1 = 3 + 1 = 4 block accesses.

Compare this to Example 3, where 12 block accesses were needed when a single-level index and binary search were used.

B-Tree and B$^+$-Tree

B-Tree and B$^+$ -Tree

A B-tree is an M-way search tree with two properties :

1. It is perfectly balanced: every leaf node is at the same depth
2. Every internal node other than the root, is at least half-full, i.e. $M/2-1 \leq \#keys \leq M-1$
3. Every internal node with k keys has $k+1$ non-null children

B^+ -Tree

- Same structure as B-trees.
- Dictionary pairs are in leaves only.
- Leaves form a doubly-linked list.
- Remaining nodes have following structure:

$$j\ a_0\ k_1\ a_1\ k_2\ a_2 \ldots k_j\ a_j$$

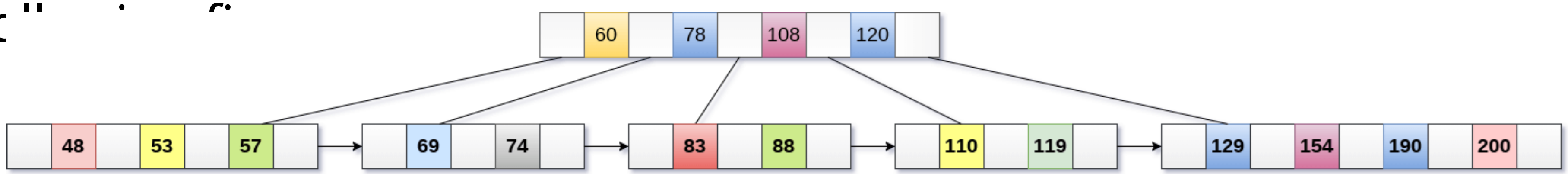
  - j = number of keys in node.
  - a_i is a pointer to a subtree.
  - $k_i <=$ smallest key in subtree a_i and $>$ largest in a_{i-1}.

B^{+} -Tree

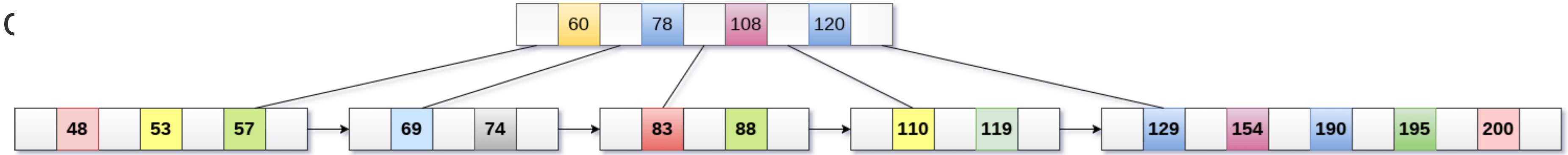
- B+ Tree is an extension of B Tree which allows efficient insertion, deletion and search operations.
- In B Tree, Keys and records both can be stored in the internal as well as leaf nodes.
- Whereas, in B+ tree, records (data) can only be stored on the leaf nodes while internal nodes can only store the key values.
- The leaf nodes of a B+ tree are linked together in the form of a singly linked lists to make the search queries more efficient.
- B+ Tree are used to store the large amount of data which can not be stored in the main memory.
- Due to the fact that, size of main memory is always limited, the internal nodes (keys to access records) of the B+ tree are stored in the main memory whereas, leaf nodes are stored in the secondary memory.

B^{+} -Tree

- Insertion in B+ Tree
- **Step 1:** Insert the new node as a leaf node
- **Step 2:** If the leaf doesn't have required space, split the node and copy the middle node to the next index node.
- **Step 3:** If the index node doesn't have required space, split the node and copy the middle element to the next index page.
- Example :
- Insert the value 195 into the B+ tree of order 5 shown in the fc

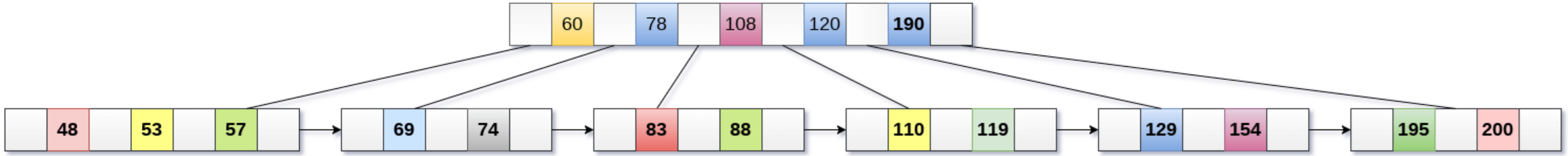

60
78
108
120
48
53
57
69
74
83
88
110
119
129
154
190
200

195 will be inserted in the right sub-tree of 120 after 190. Insert it at the desired pc


60
78
108
120
48
53
57
69
74
83
88
110
119
129
154
190
195
200

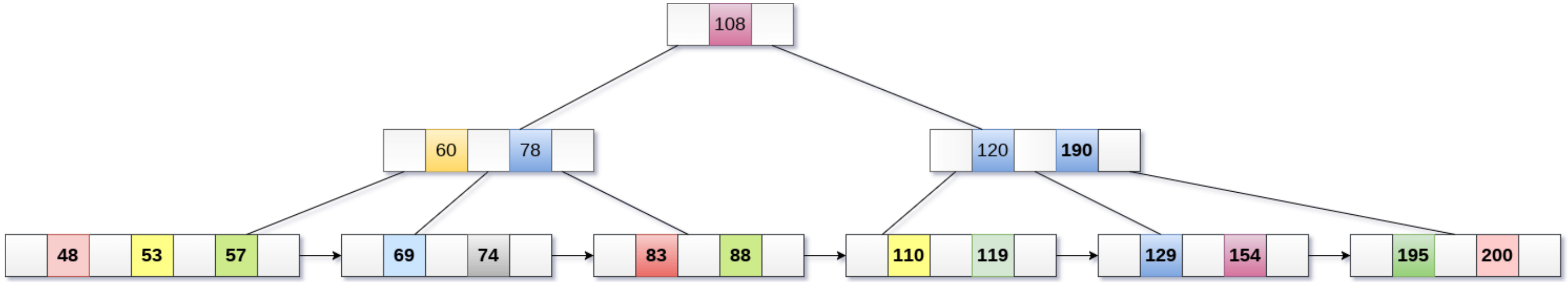
B^{+} -Tree

- The node contains greater than the maximum number of elements i.e. 4, therefore split it and place the median node up to the parent.



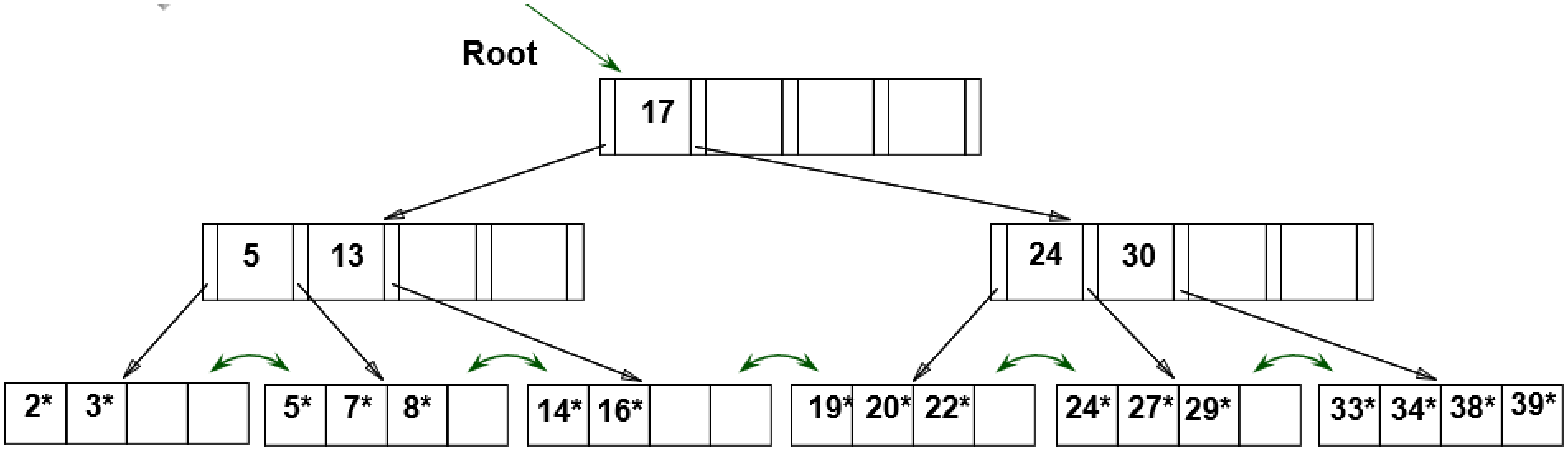
60
78
108
120
190
48
53
57
69
74
83
88
110
119
129
154
195
200

Now, the index node contains 6 children and 5 keys which violates the B+ tree properties, therefore we need to split it, shown as follows.

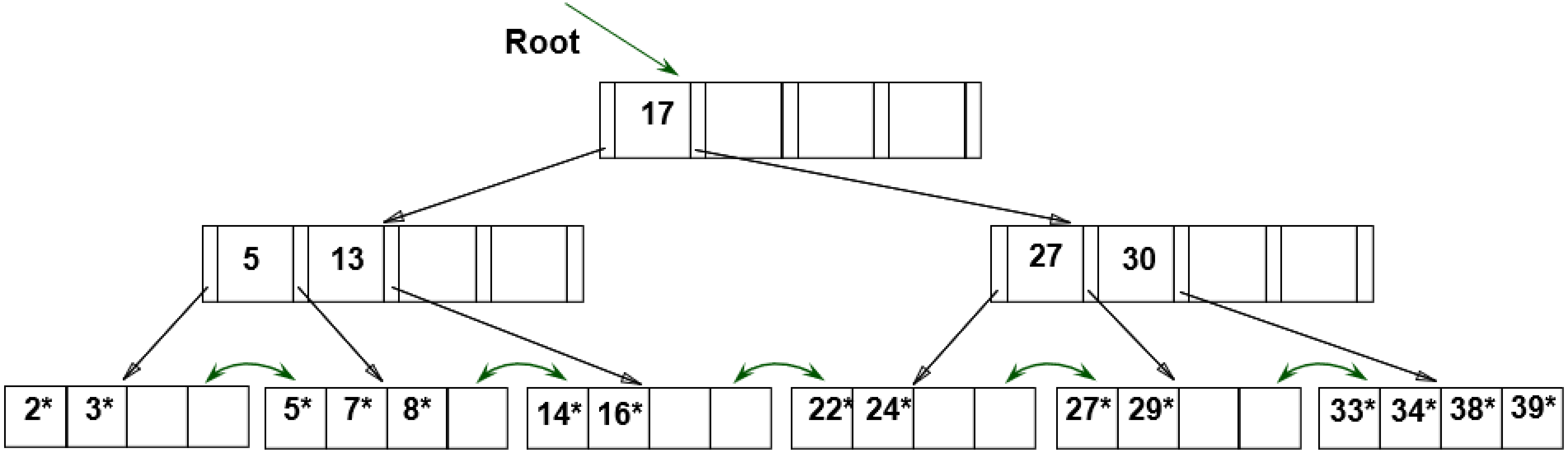


108
60
78
120
190
48
53
57
69
74
83
88
110
119
129
154
195
200

Example 2- B$^+$ Tree After Inserting 8*


Root
17
5
13
24
30
2*
3*
5*
7*
8*
14*
16*
19*
20*
22*
24*
27*
29*
33*
34*
38*
39*

Example Tree After (Inserting 8*, Then) Deleting 19* and 20* ...


Root
17
5
13
27
30
2*
3*
5*
7*
8*
14*
16*
22*
24*
27*
29*
33*
34*
38*
39*

17.3 Dynamic Multilevel Indexes Using B-Trees and B^+ -Trees

- **Tree data structure terminology**
  - Tree is formed of nodes
  - Each node (except root) has one parent and zero or more child nodes
  - Leaf node has no child nodes
    - **Unbalanced if leaf nodes occur at different levels**
  - Non-leaf node called internal node
  - Sub-tree of node consists of node and all descendant nodes

Tree Data Structure

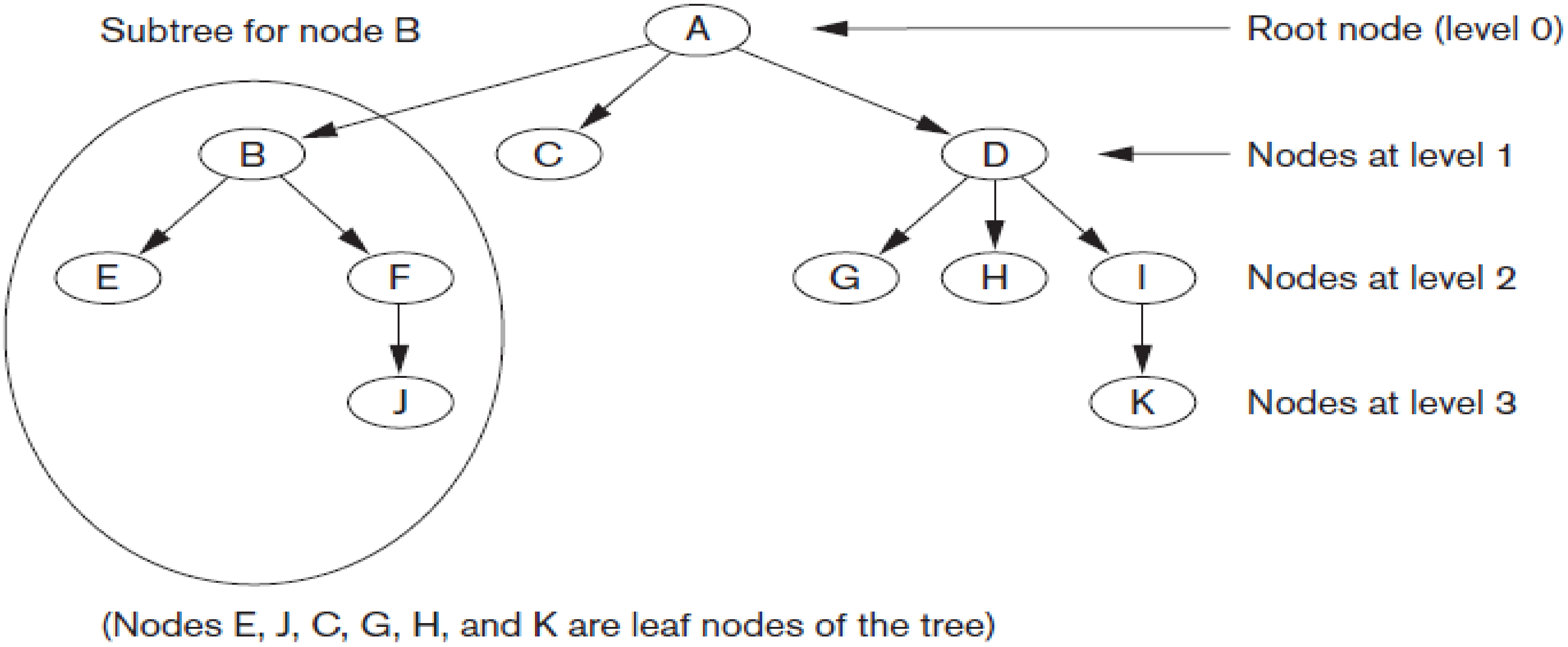

Subtree for node B
A
Root node (level 0)
B
C
D
Nodes at level 1
E
F
G
H
I
Nodes at level 2
J
K
Nodes at level 3
(Nodes E, J, C, G, H, and K are leaf nodes of the tree)

Figure 17.7 A tree data structure that shows an unbalanced tree

Search Trees and B-Trees

- **Search tree used to guide search for a record**
  - **Given value X of one of record's fields**

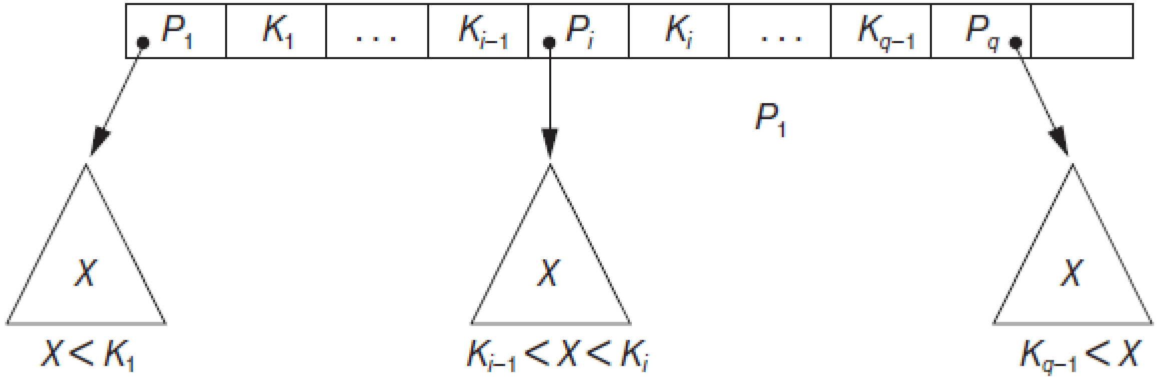

P_1
K_1
. . .
K_{i-1}
P_i
K_i
. . .
K_{q-1}
P_q
P_1
X
X
X
$X < K_1$
$K_{i-1} < X < K_i$
$K_{q-1} < X$

Figure 17.8 A node in a search tree with pointers to subtrees below it

Search Trees and B-Trees (cont'd.)

- Algorithms necessary for inserting and deleting search values into and from the tree

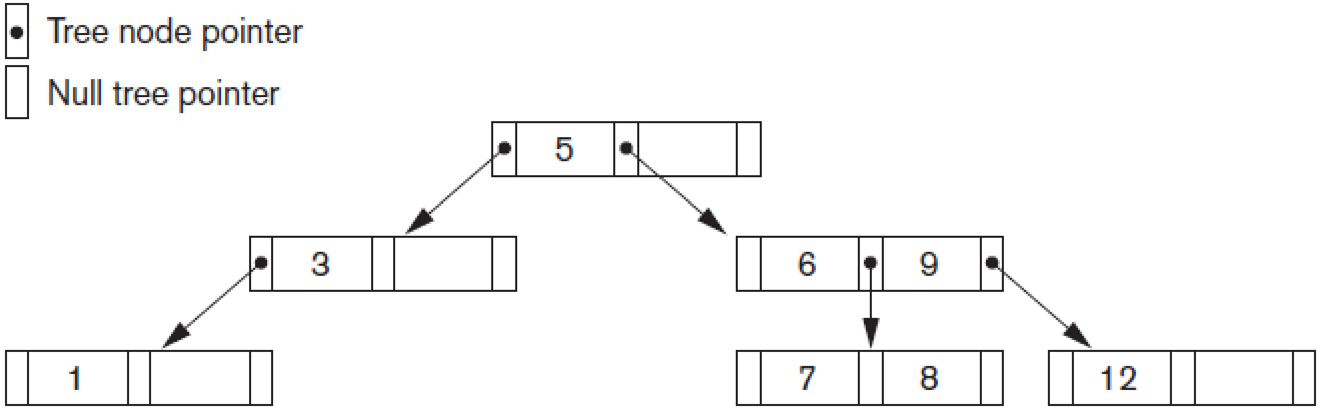
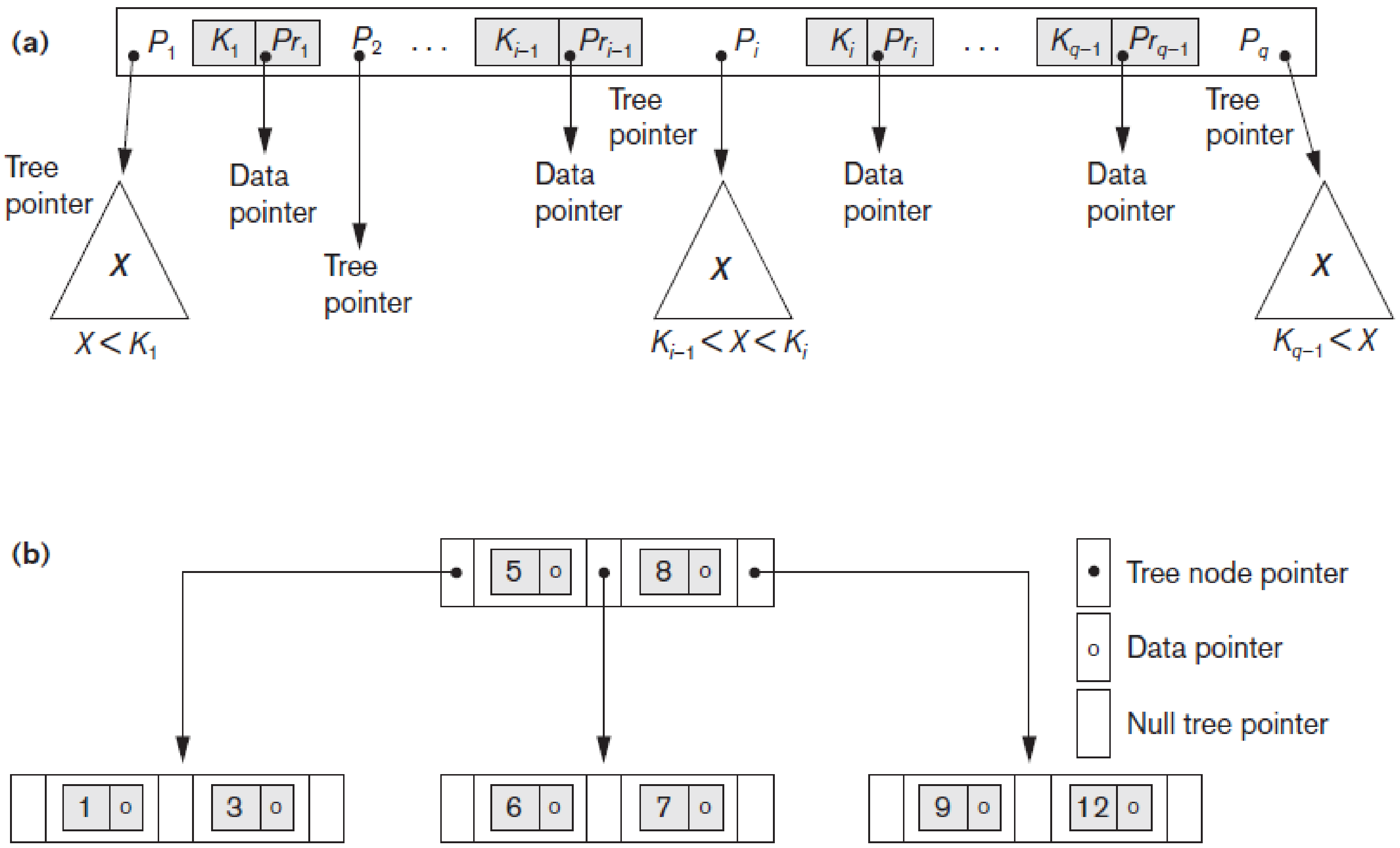

Tree node pointer
Null tree pointer
5
3
6
9
1
7
8
12

Figure 17.9 A search tree of order p = 3

B-Trees

- Provide multi-level access structure
- Tree is always balanced
- Space wasted by deletion never becomes excessive
  - Each node is at least half-full
- Each node in a B-tree of order *p* can have at most *p-1* search values

B-Tree Structures



(a)
P1
K1
Pr1
P2
Ki–1
Pri–1
Pi
Ki
Pri
Kq–1
Prq–1
Pq
Tree pointer
Data pointer
Tree pointer
Data pointer
Tree pointer
Data pointer
Data pointer
Tree pointer
X
X < K1
X
Ki–1 < X < Ki
X
Kq–1 < X
(b)
5
8
1
3
6
7
9
12
Tree node pointer
Data pointer
Null tree pointer

Figure 17.10 B-tree structures (a) A node in a B-tree with *q-1* search values (b) A B-tree of order *p=3.* The values were inserted in the order 8, 5, 1, 7, 3, 12, 9, 6

B^+ - Trees

- **Data pointers stored only at the leaf nodes**
  - Leaf nodes have an entry for every value of the search field, and a data pointer to the record if search field is a key field
  - For a non-key search field, the pointer points to a block containing pointers to the data file records
- **Internal nodes**
  - Some search field values from the leaf nodes repeated to guide search

B^+ -Trees (cont’d.)

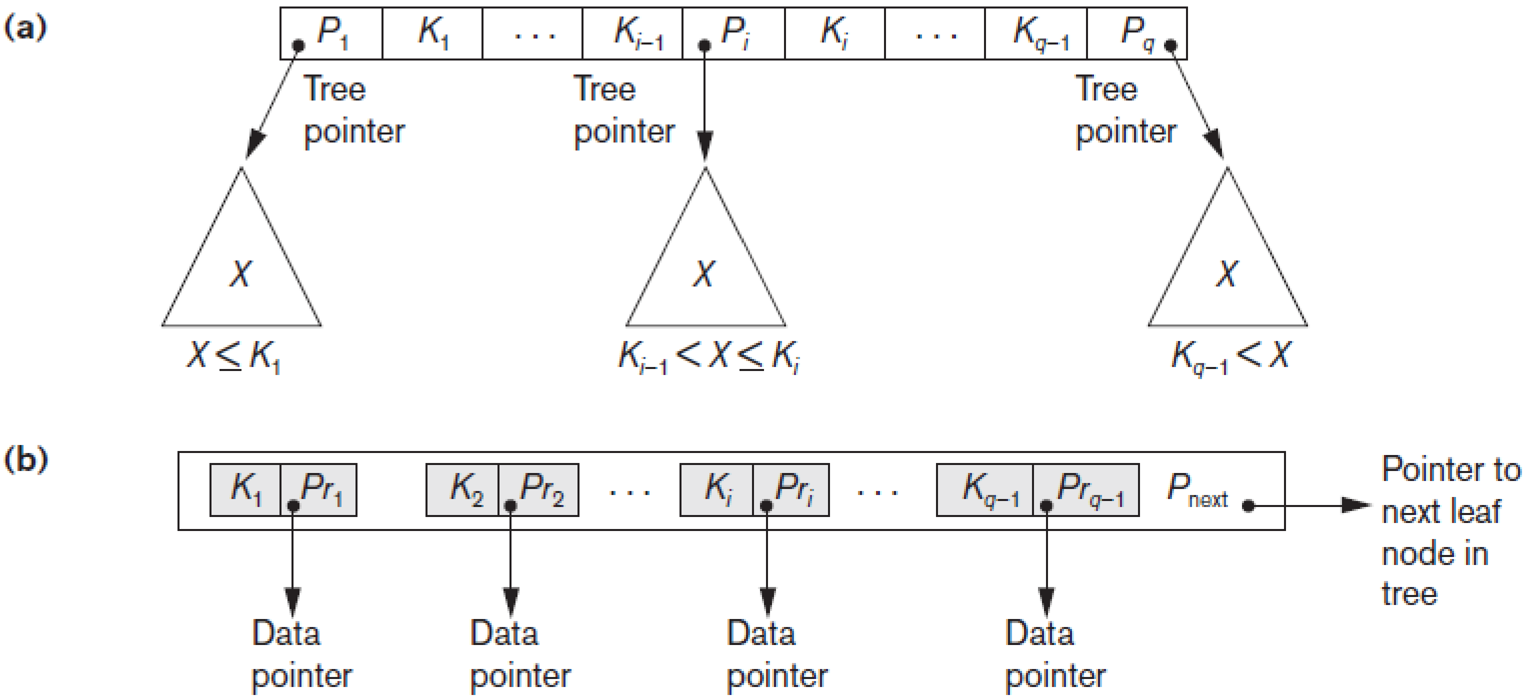
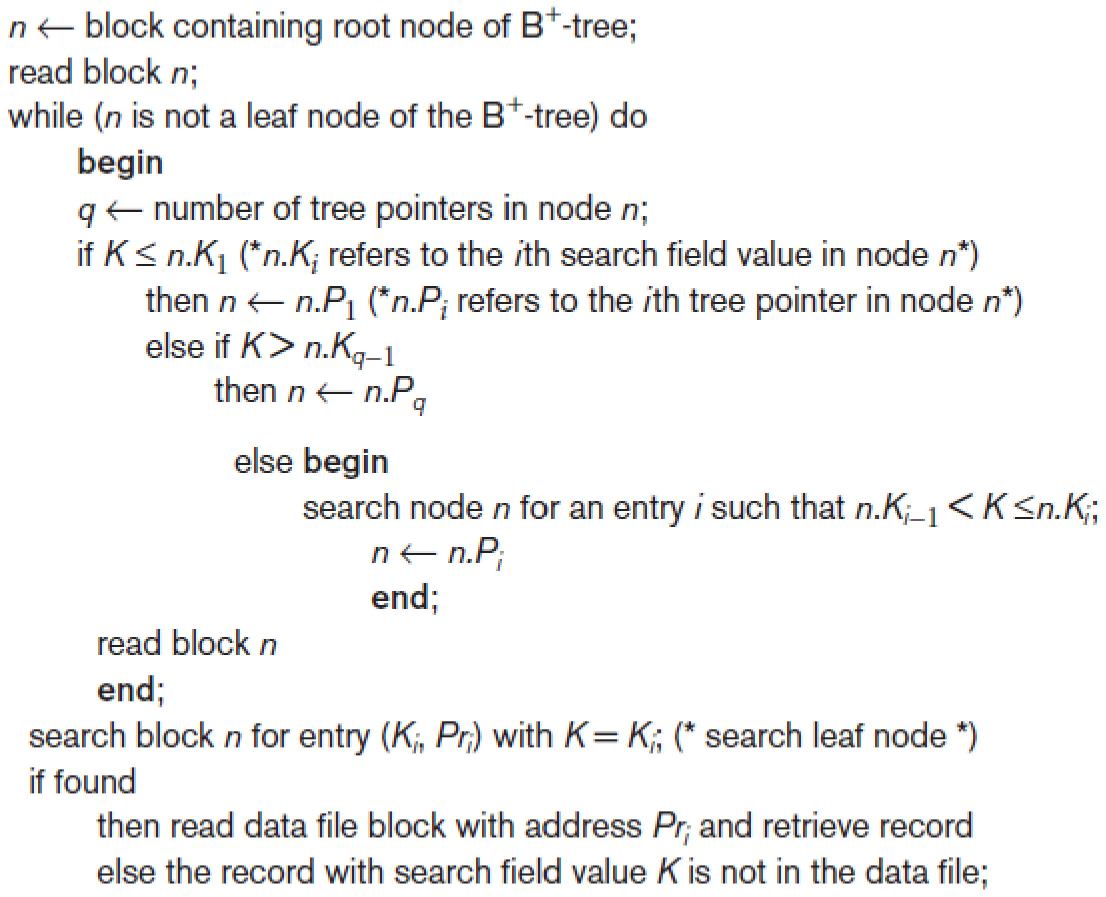

(a)
P_1
K_1
. . .
K_{i-1}
P_i
K_i
. . .
K_{q-1}
P_q
Tree pointer
Tree pointer
Tree pointer
X
X
X
$X \leq K_1$
$K_{i-1} < X \leq K_i$
$K_{q-1} < X$
(b)
K_1
Pr_1
K_2
Pr_2
. . .
K_i
Pr_i
. . .
K_{q-1}
Pr_{q-1}
P_{next}
Pointer to next leaf node in tree
Data pointer
Data pointer
Data pointer
Data pointer

Figure 17.11 The nodes of a B+-tree (a) Internal node of a B+-tree with *q-1* search values (b) Leaf node of a B+ -tree with *q-1* search values and *q-1* data pointers

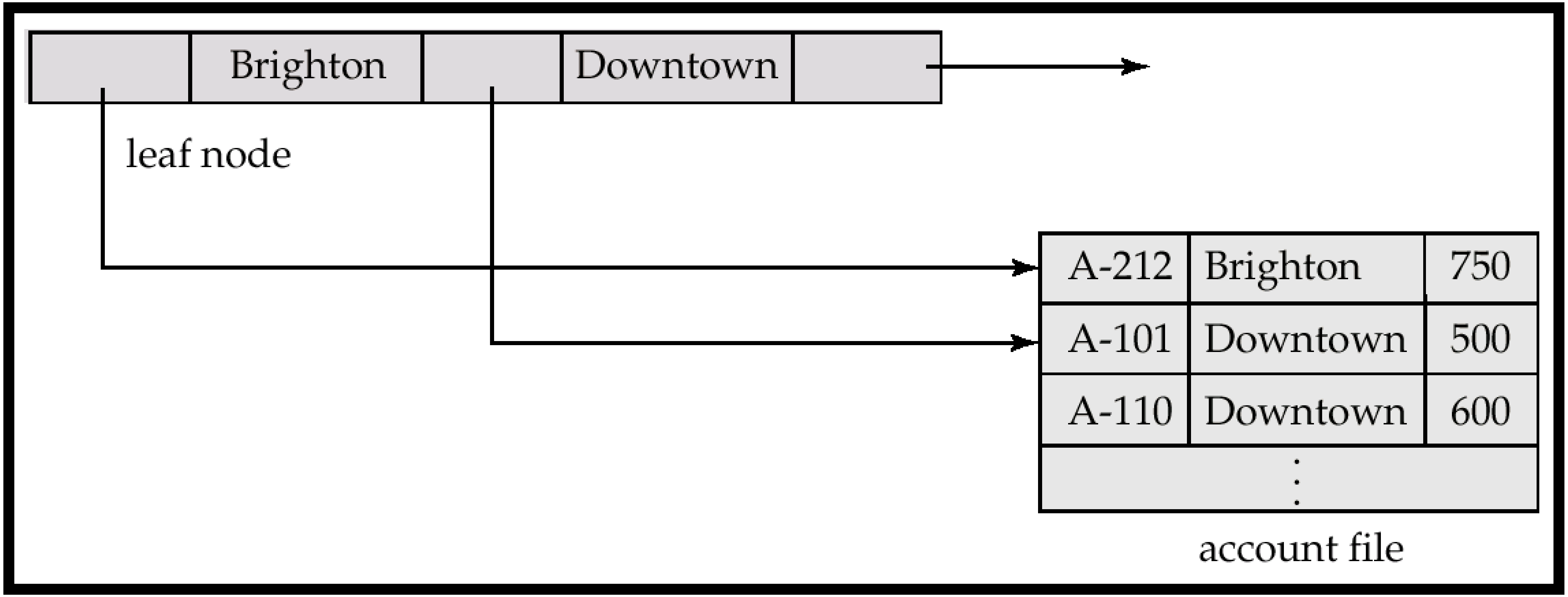
Searching for a Record With Search Key Field Value K, Using a B^+ -Tree



```
$n \leftarrow$ block containing root node of $B^+$-tree;
read block $n$;
while ($n$ is not a leaf node of the $B^+$-tree) do
    begin
    $q \leftarrow$ number of tree pointers in node $n$;
    if $K \leq n.K_1$ (*$n.K_i$ refers to the $i$th search field value in node $n$*)
        then $n \leftarrow n.P_1$ (*$n.P_i$ refers to the $i$th tree pointer in node $n$*)
        else if $K > n.K_{q-1}$
            then $n \leftarrow n.P_q$
            else begin
                search node $n$ for an entry $i$ such that $n.K_{i-1} < K \leq n.K_i$;
                    $n \leftarrow n.P_i$
                    end;
     read block $n$
     end;
 search block $n$ for entry ($K_i$, $Pr_i$) with $K = K_i$; (* search leaf node *)
 if found
     then read data file block with address $Pr_i$ and retrieve record
     else the record with search field value $K$ is not in the data file;
```

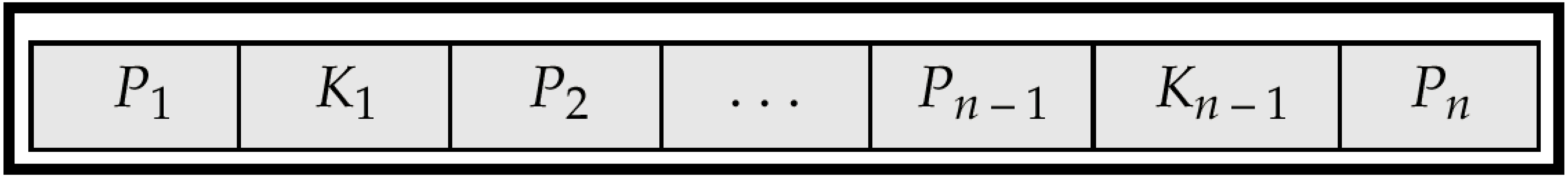
Algorithm 17.2 Searching for a record with search key field value K, using a B+ -Tree

Leaf Nodes in B$^+$-Trees


Brighton
Downtown
leaf node
A-212
Brighton
750
A-101
Downtown
500
A-110
Downtown
600
account file

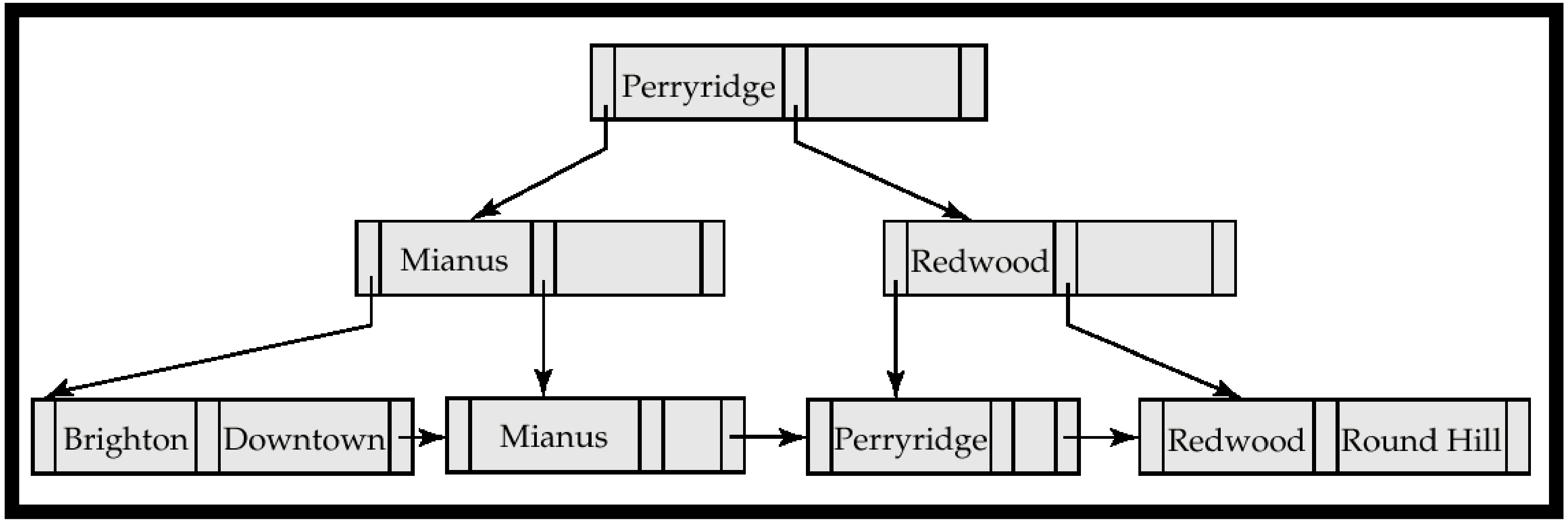
Non-Leaf Nodes in B$^+$-Trees

- **All the search-keys in the subtree to which P_1 points are less than K_1**
- **For $2 \leq i \leq n - 1$, all the search-keys in the subtree to which P_i points have values greater than or equal to K_{i-1} and less than K_{m-1}**



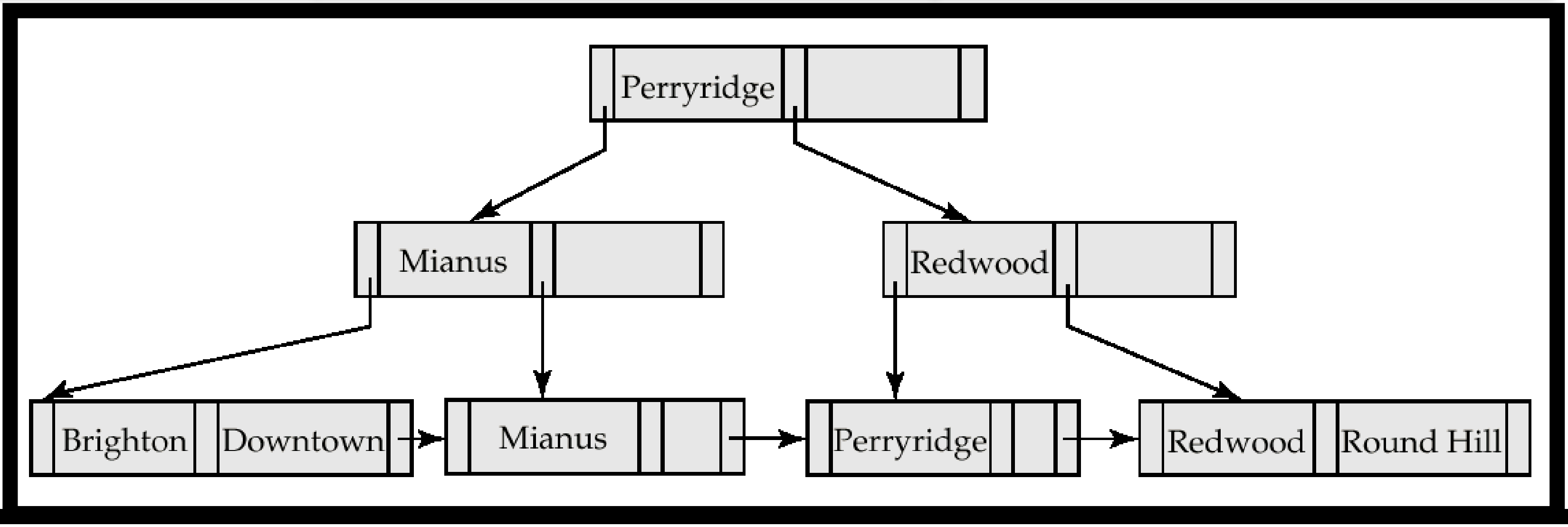
P_1	K_1	P_2	$\ldots$	P_{n-1}	K_{n-1}	P_n

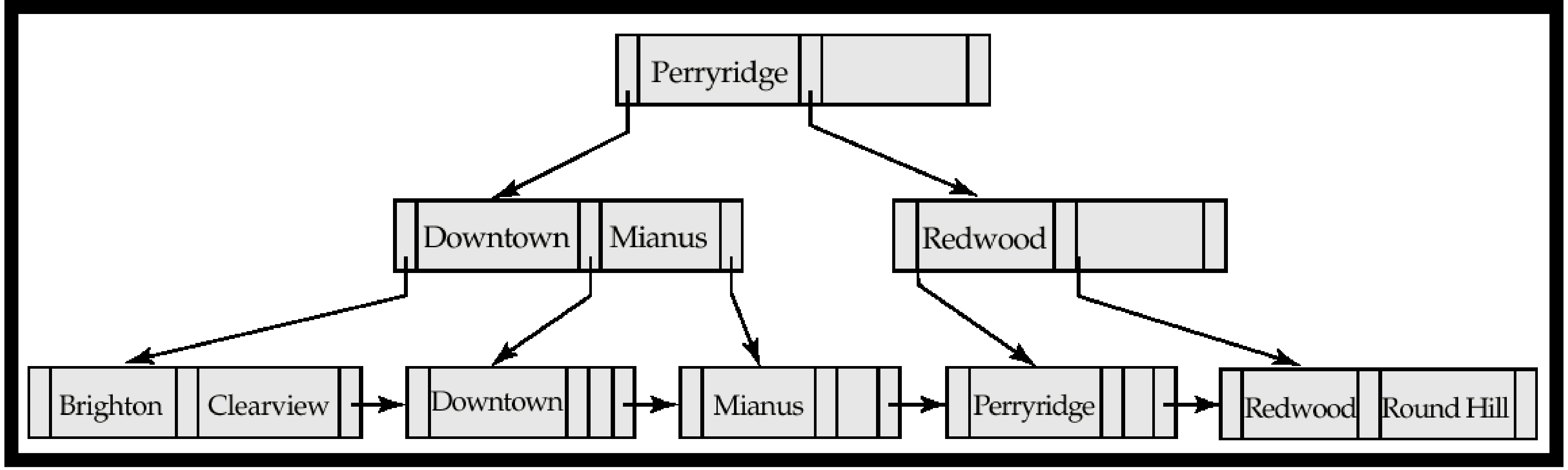
Example of a B$^+$-tree


Perryridge
Mianus
Redwood
Brighton
Downtown
Mianus
Perryridge
Redwood
Round Hill

B$^+$-tree for account file (n = 3)

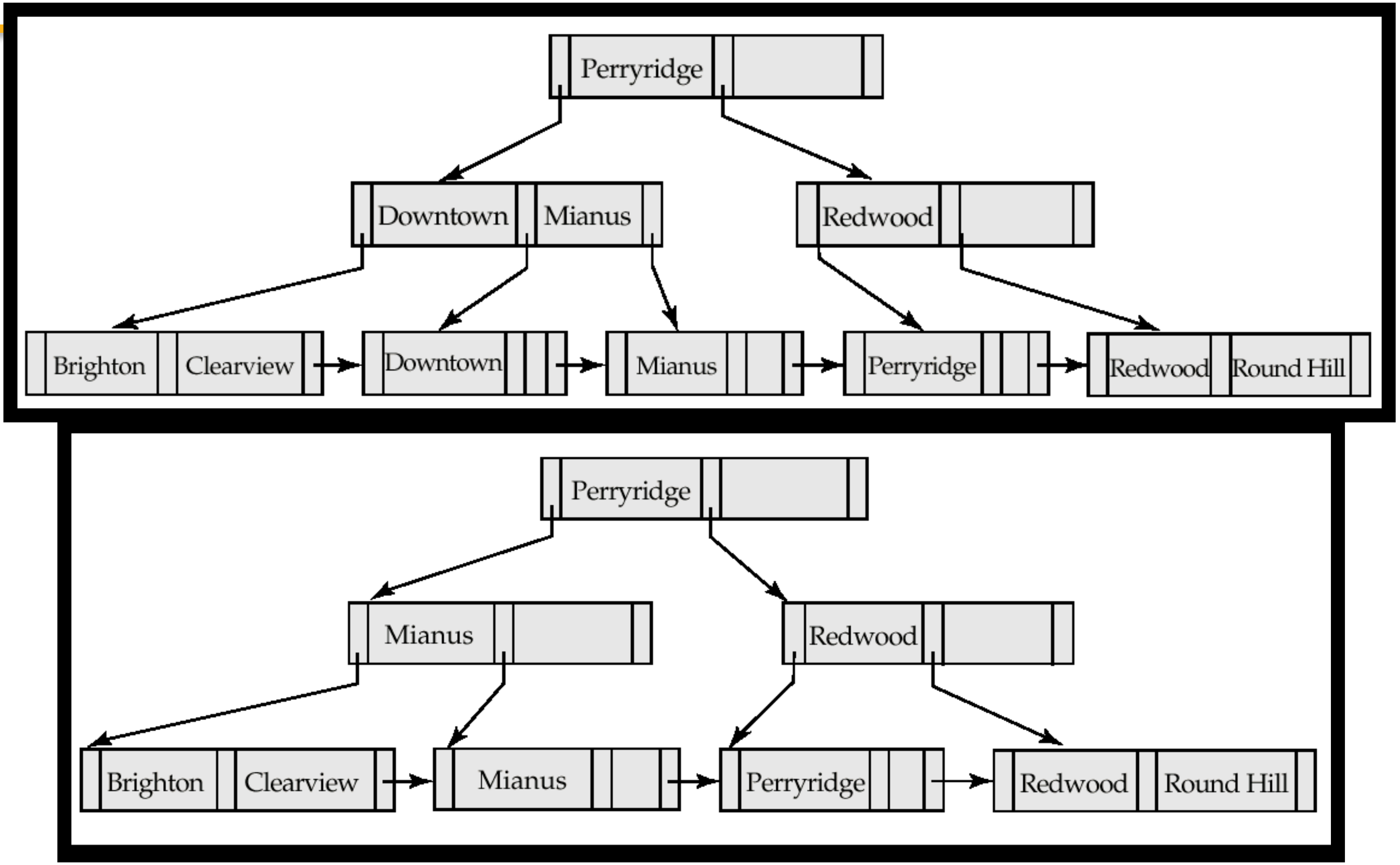
Updates on B$^+$-Trees: Insertion


Perryridge
Mianus
Redwood
Brighton
Downtown
Mianus
Perryridge
Redwood
Round Hill


Perryridge
Downtown
Mianus
Redwood
Brighton
Clearview
Downtown
Mianus
Perryridge
Redwood
Round Hill

B$^+$-Tree before and after insertion of "Clearview"

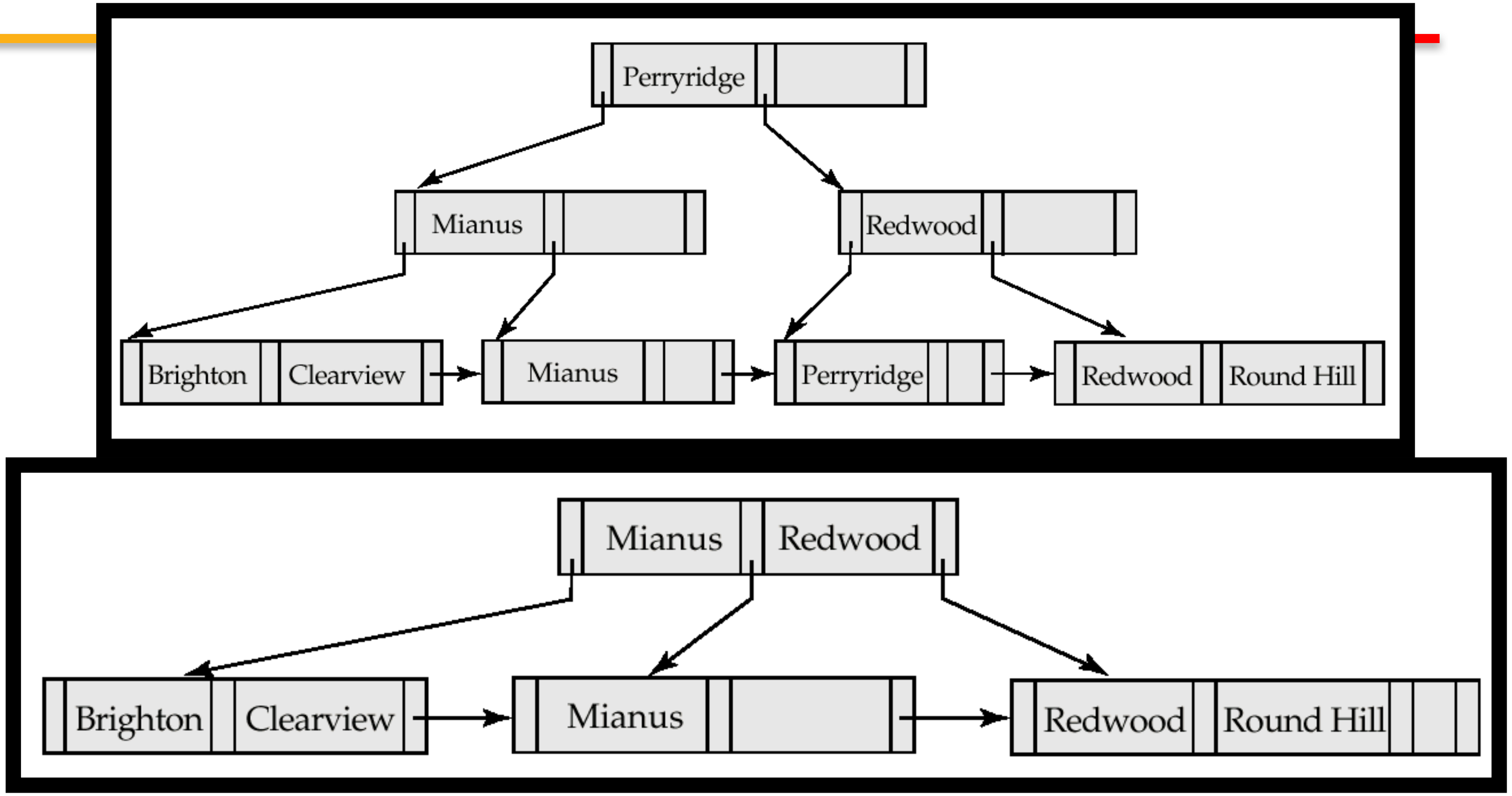
Examples of B$^+$-Tree Deletion


Perryridge
Downtown
Mianus
Redwood
Brighton
Clearview
Downtown
Mianus
Perryridge
Redwood
Round Hill
Perryridge
Mianus
Redwood
Brighton
Clearview
Mianus
Perryridge
Redwood
Round Hill

Before and after deleting “Downtown”

- The removal of the leaf node containing “Downtown” did not result in its parent having too little pointers. So the cascaded deletions stopped with the deleted leaf node’s parent.

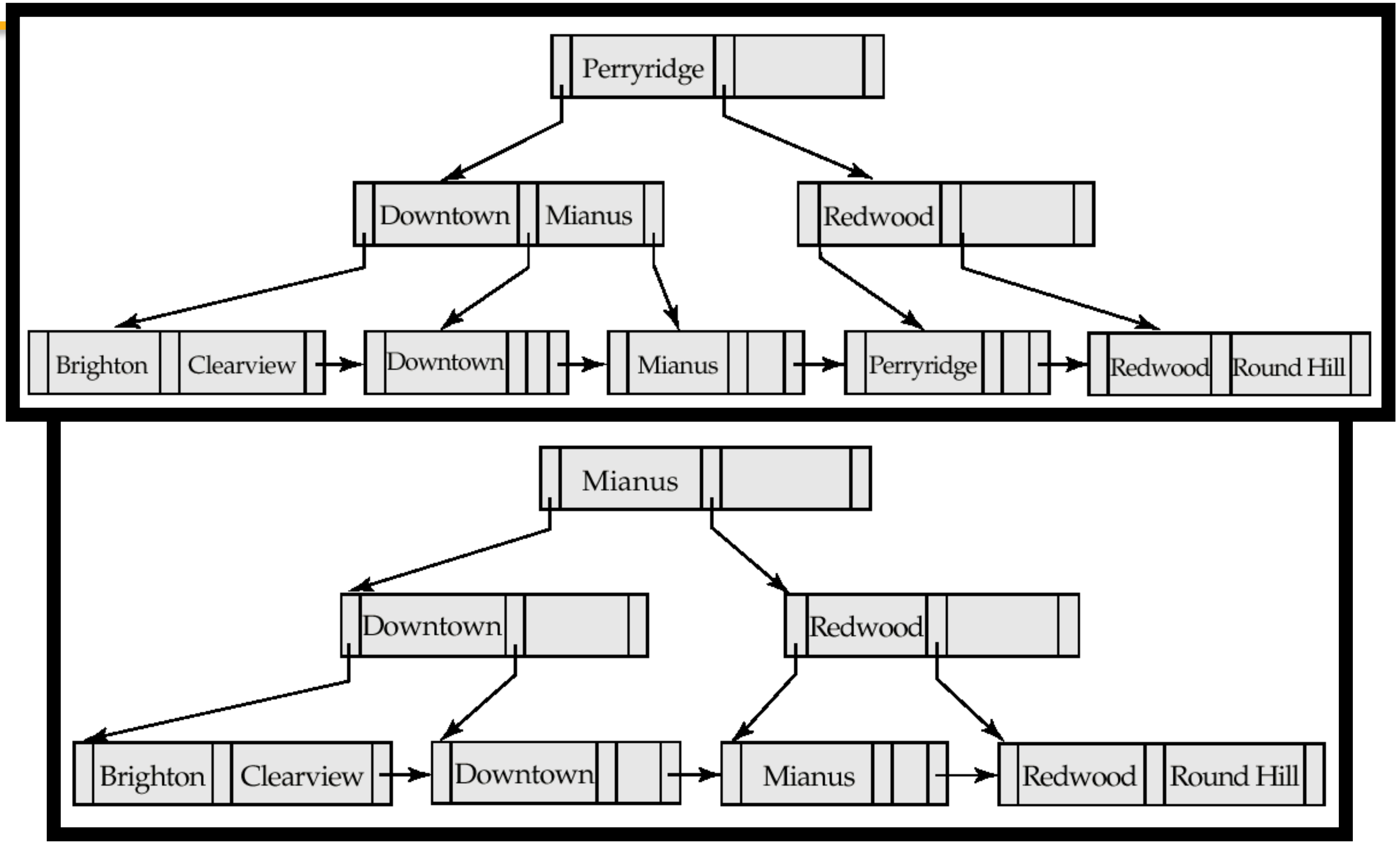
Examples of B$^+$-Tree Deletion


Perryridge
Mianus
Redwood
Brighton
Clearview
Mianus
Perryridge
Redwood
Round Hill
Mianus
Redwood
Brighton
Clearview
Mianus
Redwood
Round Hill

Deletion of “Perryridge” from result of previous example

- Node with “Perryridge” becomes underfull (actually empty, in this special case) and merged with its sibling.
- As a result “Perryridge” node’s parent became underfull, and was merged with its sibling (and an entry was deleted from their parent)
- Root node then had only one child, and was deleted and its child became the new root node

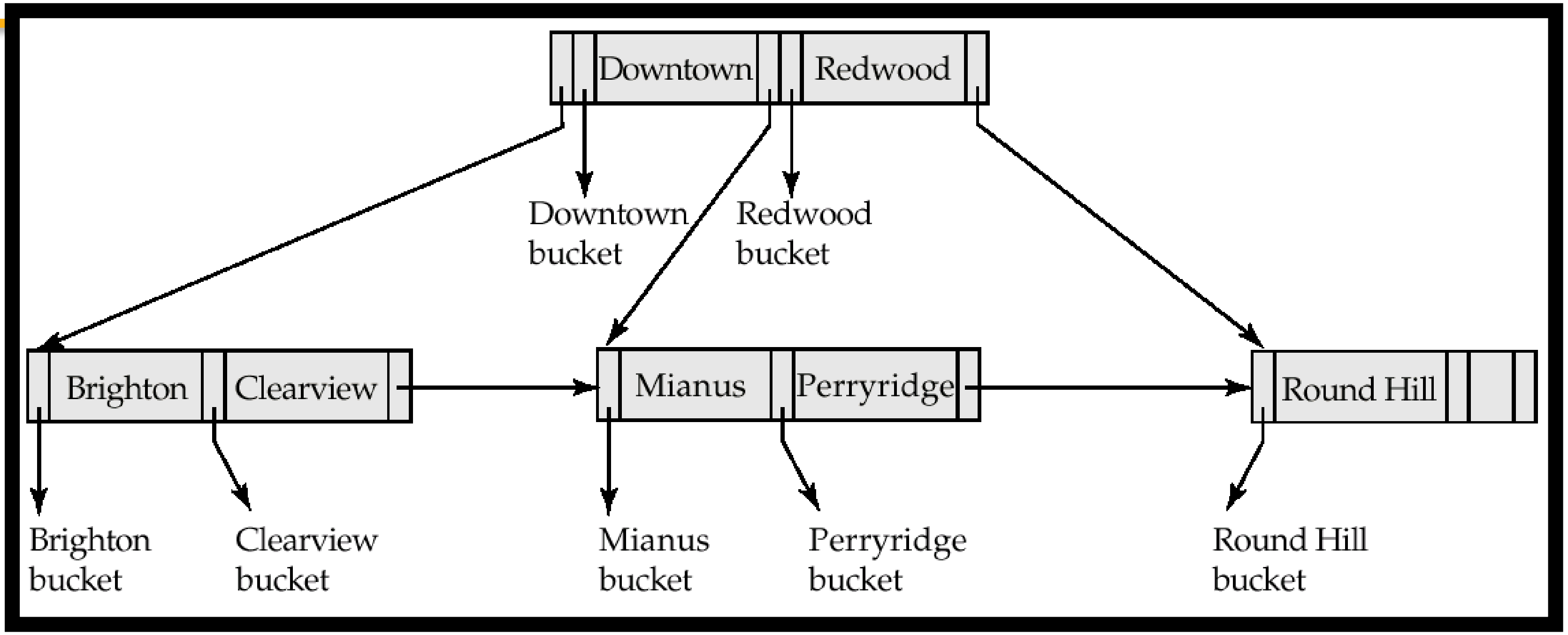
Example of B^+-tree Deletion


Perryridge
Downtown
Mianus
Redwood
Brighton
Clearview
Downtown
Mianus
Perryridge
Redwood
Round Hill
Mianus
Downtown
Redwood
Brighton
Clearview
Downtown
Mianus
Redwood
Round Hill

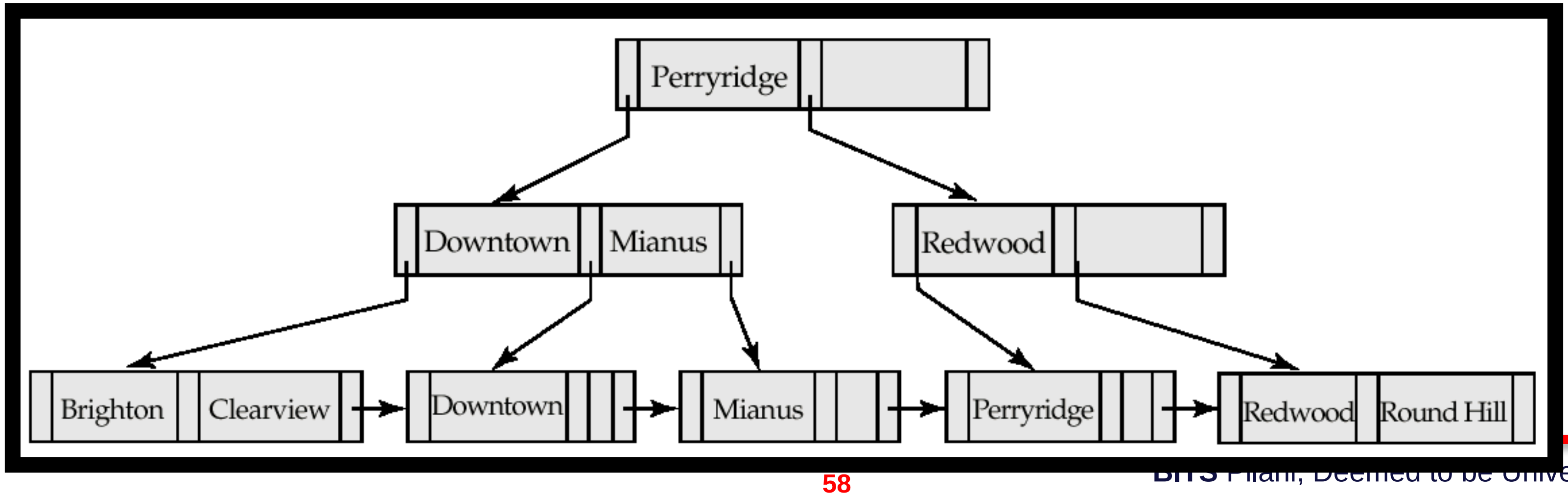
Before and after deletion of “Perryridge” from earlier example

- Parent of leaf containing Perryridge became underfull, and borrowed a pointer from its left sibling
- Search-key value in the parent’s parent changes as a result

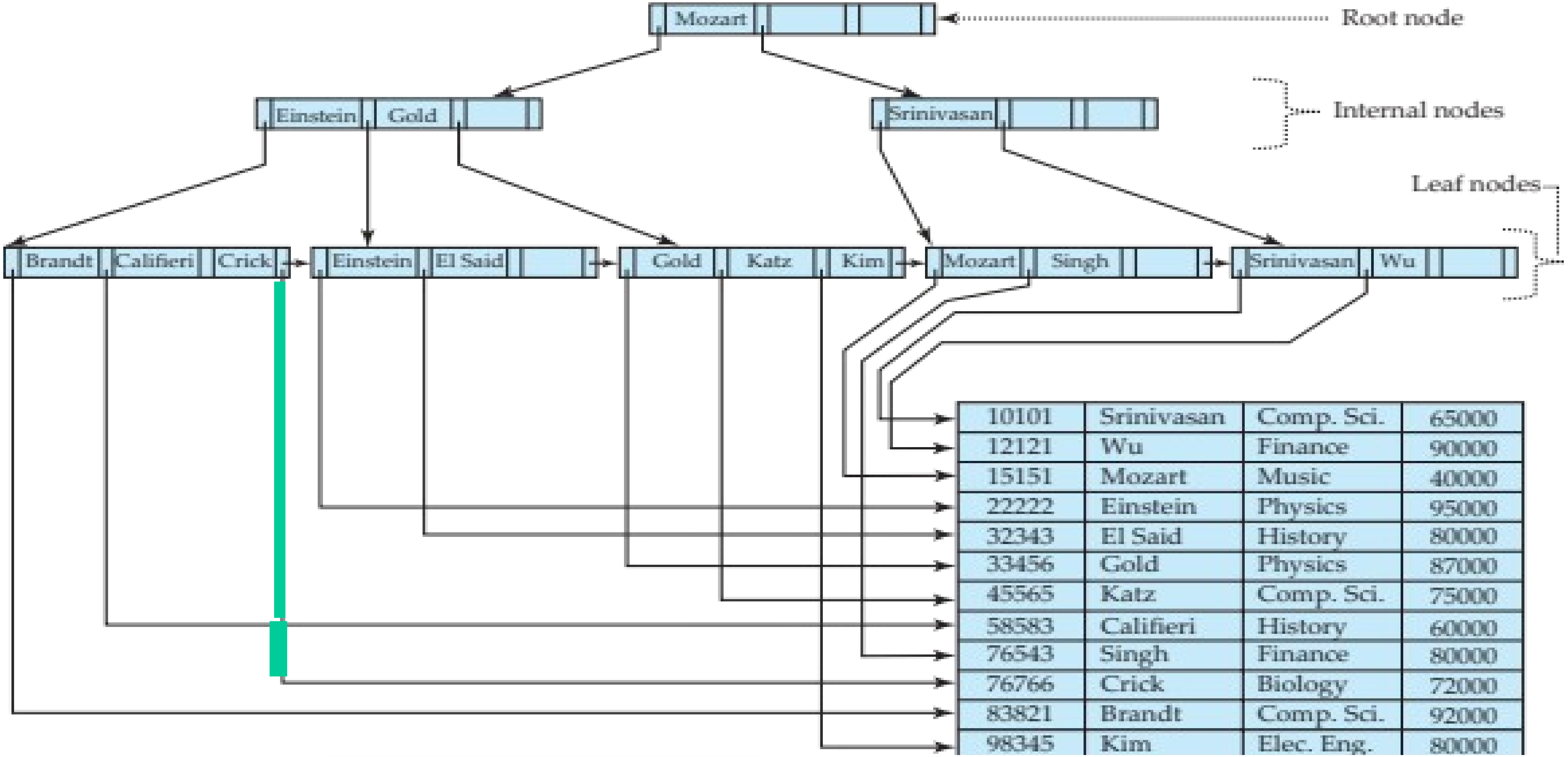
B-Tree Index File Example


Downtown
Redwood
Downtown bucket
Redwood bucket
Brighton
Clearview
Mianus
Perryridge
Round Hill
Brighton bucket
Clearview bucket
Mianus bucket
Perryridge bucket
Round Hill bucket

B-tree (above) and B+-tree (below) on same data

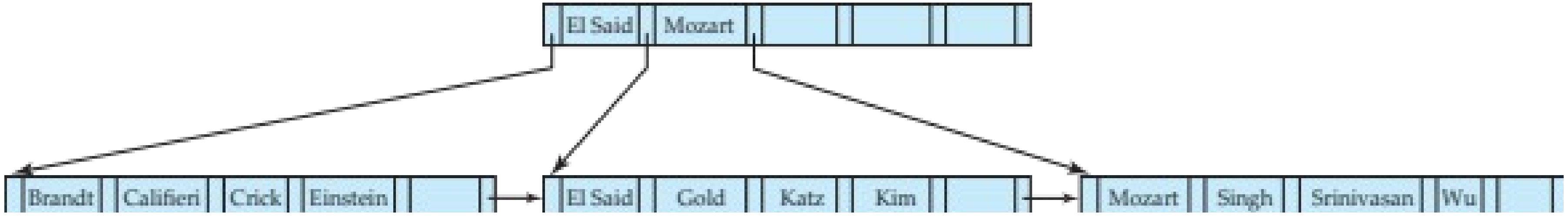

Perryridge
Downtown
Mianus
Redwood
Brighton
Clearview
Downtown
Mianus
Perryridge
Redwood
Round Hill

Example of B$^+$-Tree


Mozart
Root node
Einstein
Gold
Srinivasan
Internal nodes
Leaf nodes
Brandt
Califieri
Crick
Einstein
El Said
Gold
Katz
Kim
Mozart
Singh
Srinivasan
Wu

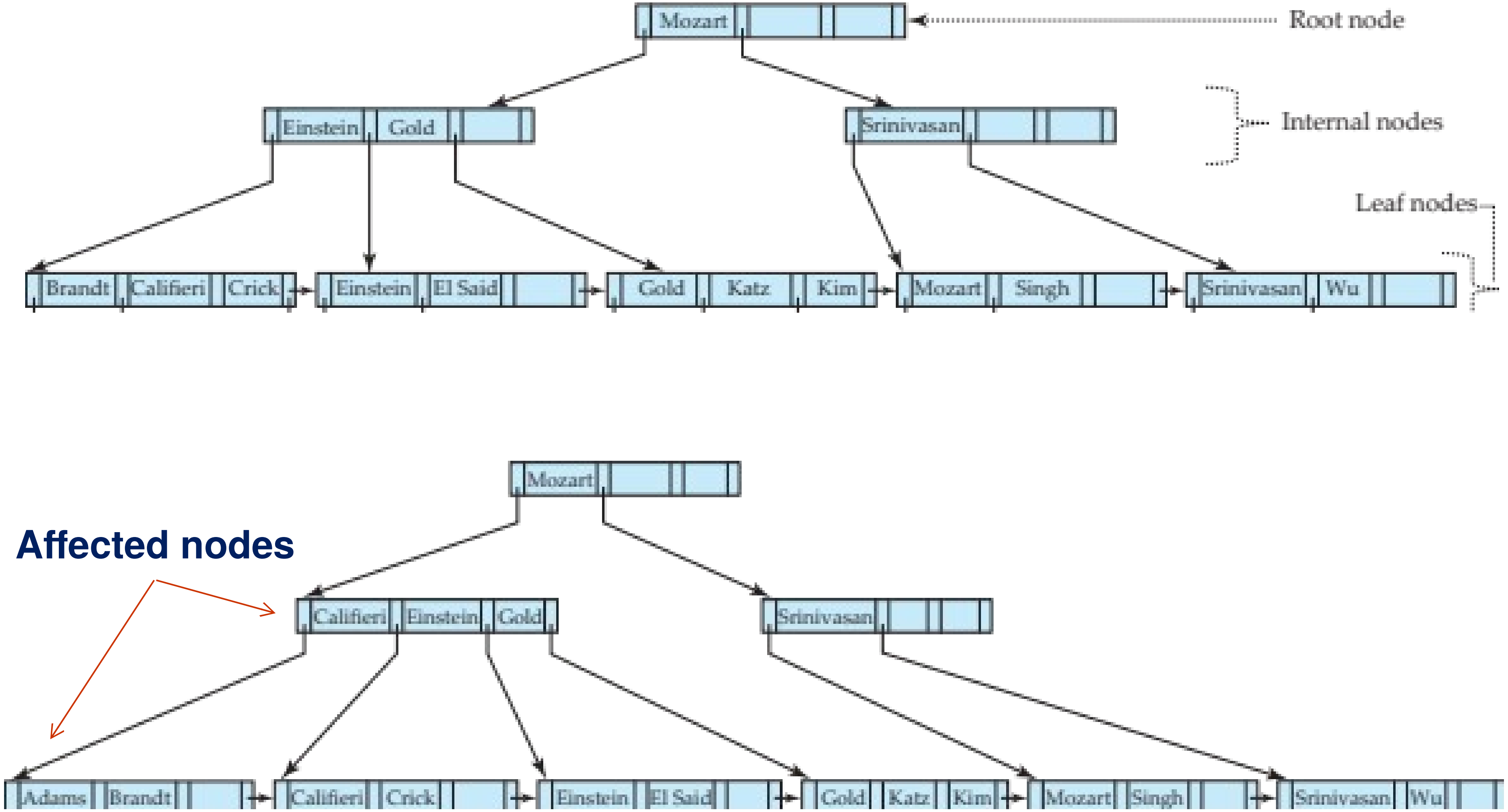
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	80000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	60000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Example of B⁺-tree


El Said
Mozart
Brandt
Califieri
Crick
Einstein
El Said
Gold
Katz
Kim
Mozart
Singh
Srinivasan
Wu

B⁺-tree for *instructor* file (*n* = 6)

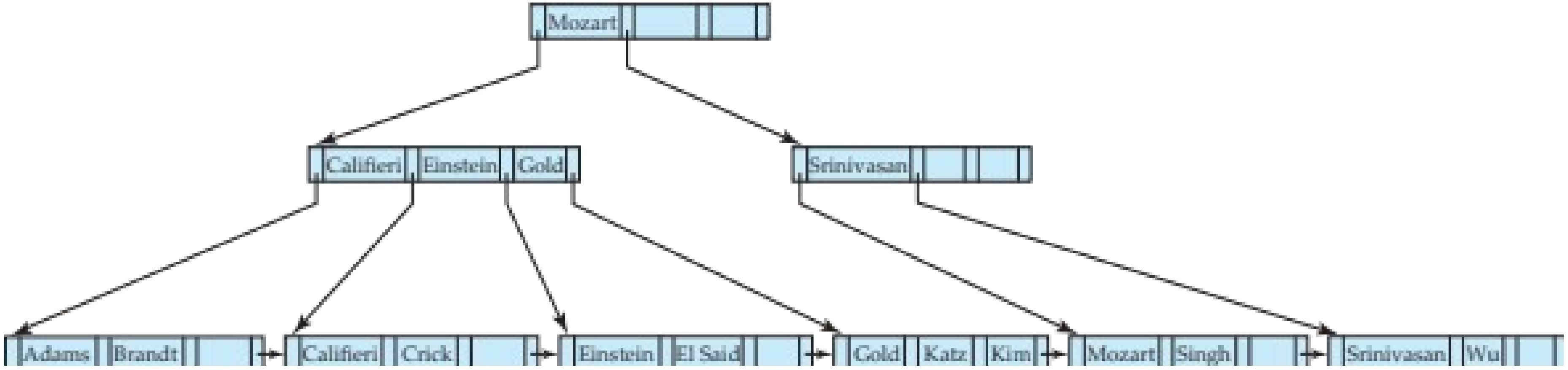
B$^+$-Tree Insertion


Mozart
Root node
Einstein
Gold
Srinivasan
Internal nodes
Leaf nodes
Brandt
Califieri
Crick
Einstein
El Said
Gold
Katz
Kim
Mozart
Singh
Srinivasan
Wu
Mozart
Califieri
Einstein
Gold
Srinivasan
Adams
Brandt
Califieri
Crick
Einstein
El Said
Gold
Katz
Kim
Mozart
Singh
Srinivasan
Wu

Affected nodes

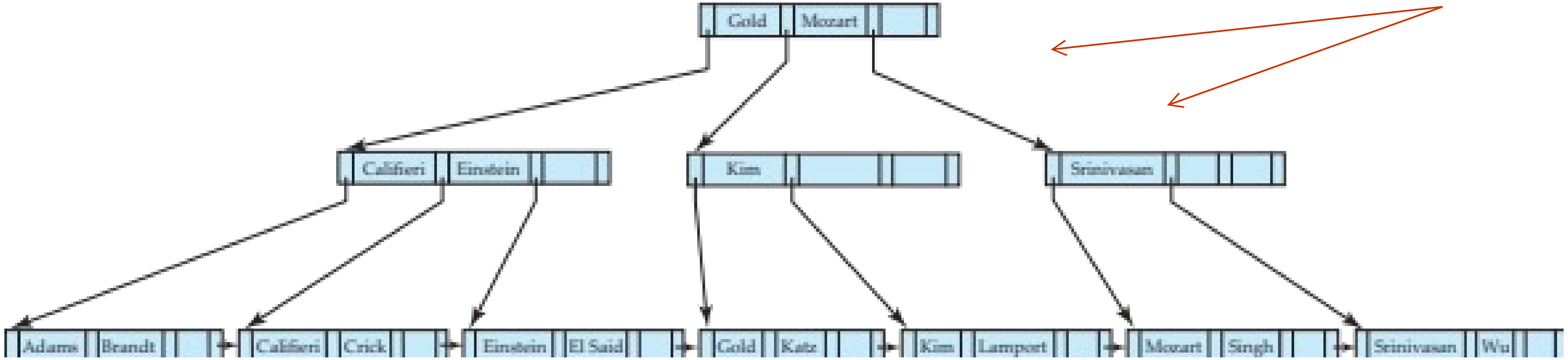
B$^+$-Tree before and after insertion of “Adams”

B^+-Tree Insertion


Mozart
Califieri
Einstein
Gold
Srinivasan
Adams
Brandt
Califieri
Crick
Einstein
El Said
Gold
Katz
Kim
Mozart
Singh
Srinivasan
Wu

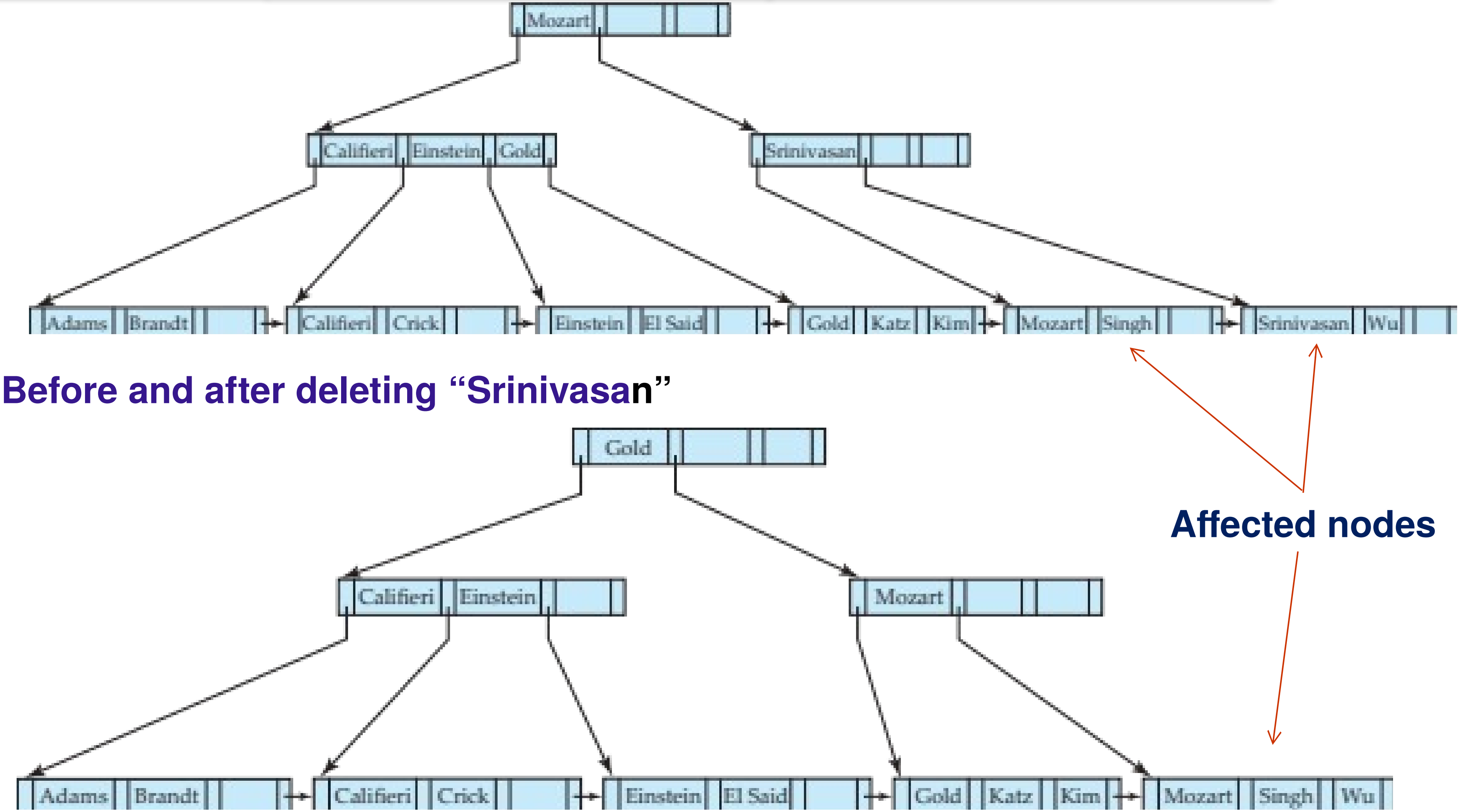
B^+-Tree before and after insertion of "Lamport"

Affected nodes


Gold
Mozart
Califieri
Einstein
Kim
Srinivasan
Adams
Brandt
Califieri
Crick
Einstein
El Said
Gold
Katz
Kim
Lamport
Mozart
Singh
Srinivasan
Wu

Affected nodes

Examples of B⁺-Tree Deletion

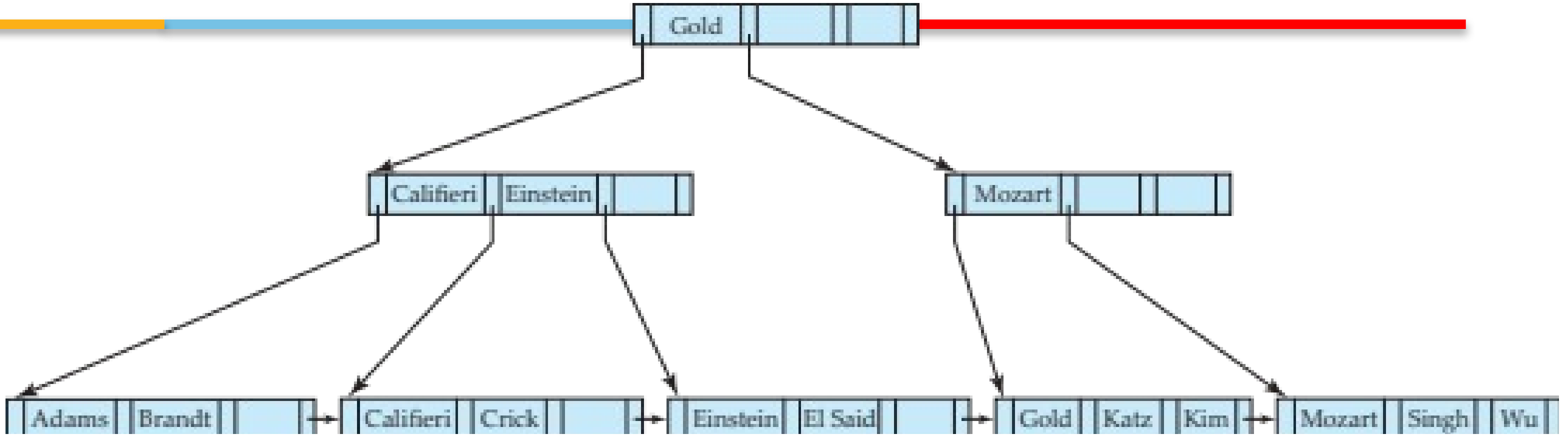

Mozart
Califieri
Einstein
Gold
Srinivasan
Adams
Brandt
Califieri
Crick
Einstein
El Said
Gold
Katz
Kim
Mozart
Singh
Srinivasan
Wu
Gold
Califieri
Einstein
Mozart
Adams
Brandt
Califieri
Crick
Einstein
El Said
Gold
Katz
Kim
Mozart
Singh
Wu

Before and after deleting “Srinivasan”

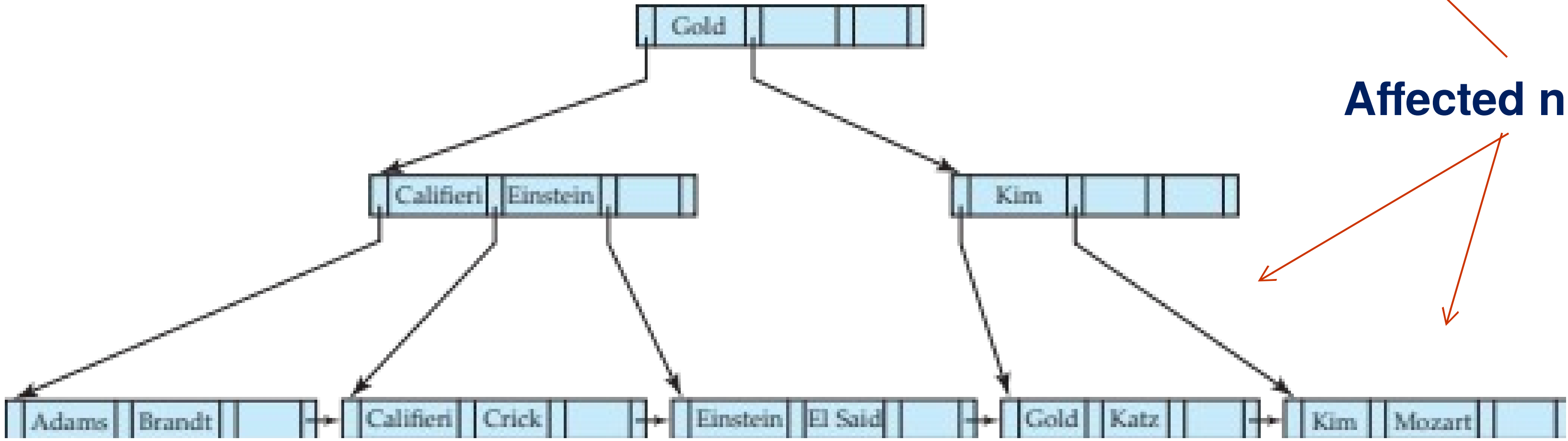
Affected nodes

- Deleting “Srinivasan” causes **merging** of under-full leaves

Examples of B$^+$-Tree Deletion


Gold
Califieri
Einstein
Mozart
Adams
Brandt
Califieri
Crick
Einstein
El Said
Gold
Katz
Kim
Mozart
Singh
Wu

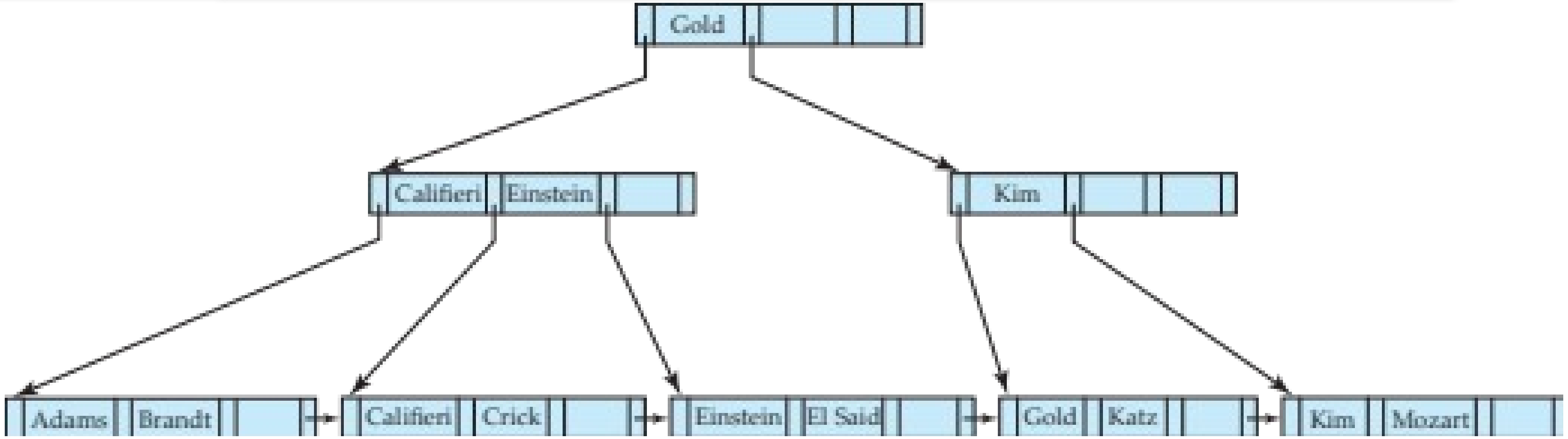
Before and after deleting "Singh" and "Wu"


Gold
Califieri
Einstein
Kim
Adams
Brandt
Califieri
Crick
Einstein
El Said
Gold
Katz
Kim
Mozart

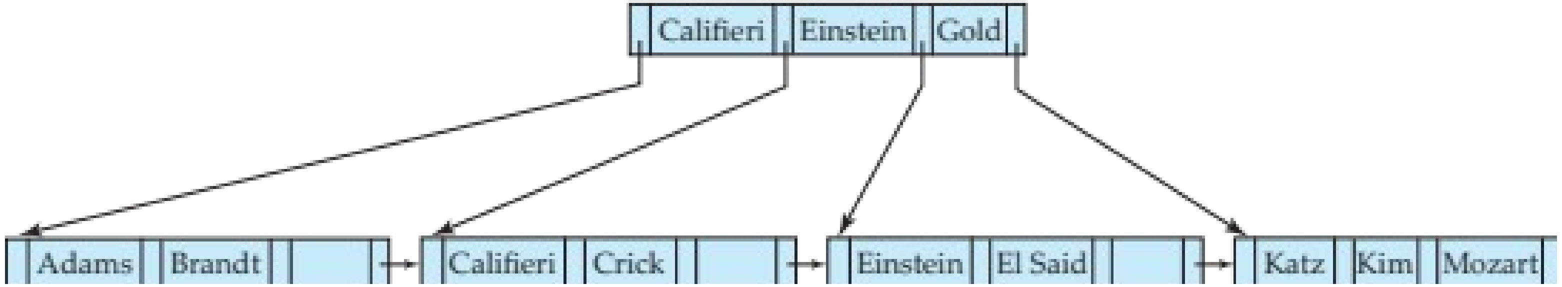
Affected nodes

- Leaf containing Singh and Wu became underfull, and **borrowed a value** Kim from its left sibling
- Search-key value in the parent changes as a result

Example of B^+-tree Deletion (Cont.)

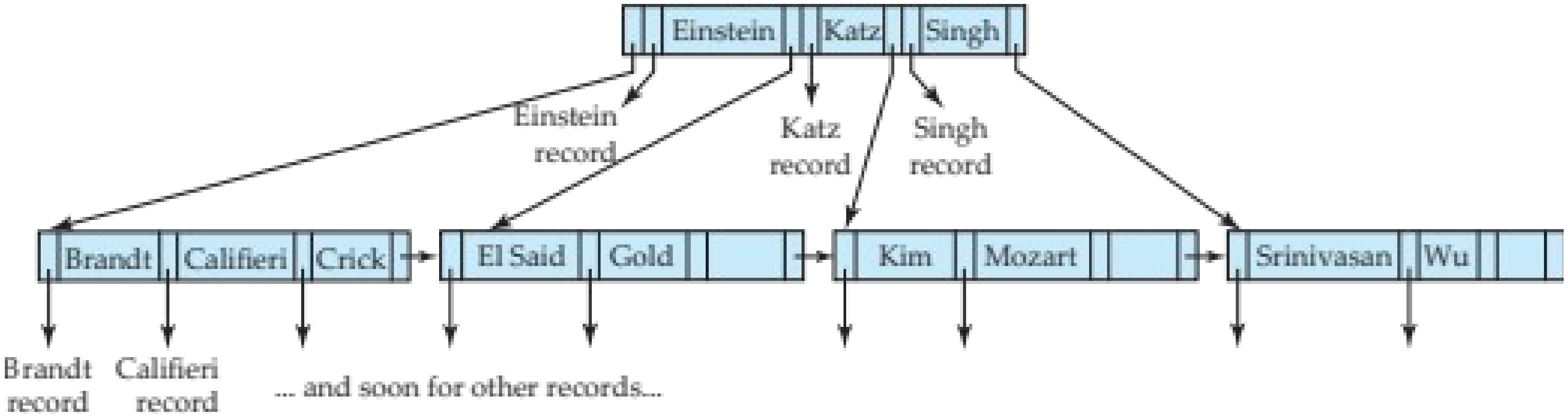

Gold
Califieri
Einstein
Kim
Adams
Brandt
Califieri
Crick
Einstein
El Said
Gold
Katz
Kim
Mozart

Before and after deletion of “Gold”

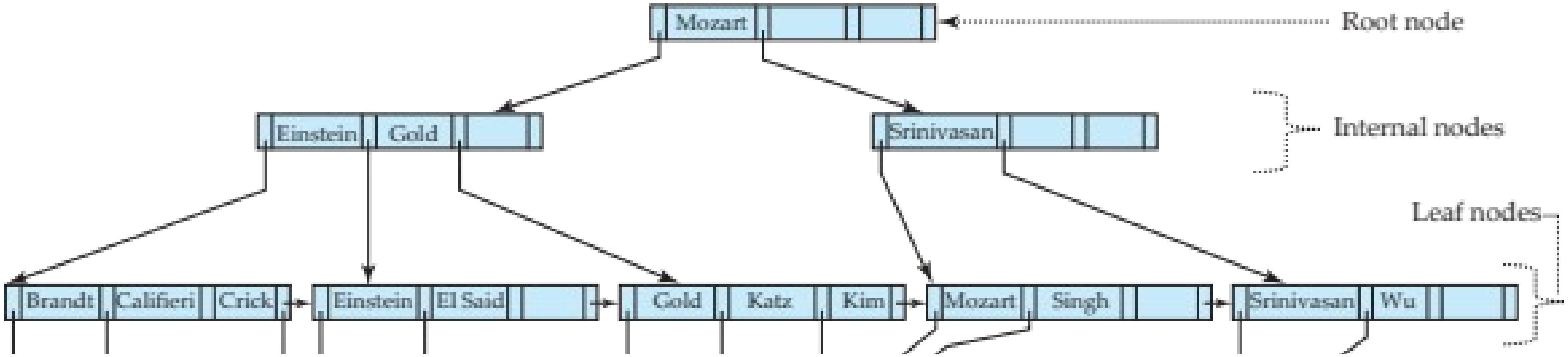

Califieri
Einstein
Gold
Adams
Brandt
Califieri
Crick
Einstein
El Said
Katz
Kim
Mozart

- Node with Gold and Katz became underfull, and was merged with its sibling
- Parent node becomes underfull, and is merged with its sibling
  - Value separating two nodes (at the parent) is pulled down when merging
- Root node then has only one child, and is deleted

B-Tree Index File Example

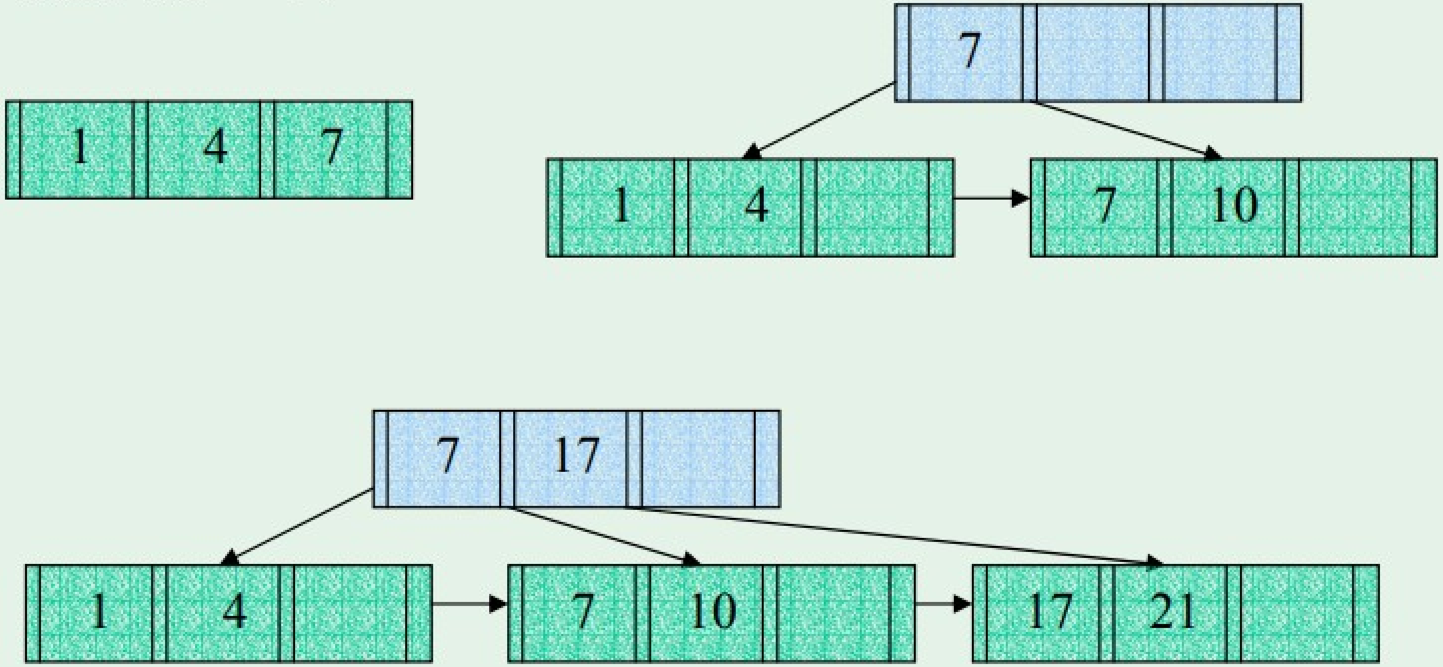

Einstein
Katz
Singh
Einstein record
Katz record
Singh record
Brandt
Califieri
Crick
El Said
Gold
Kim
Mozart
Srinivasan
Wu
Brandt record
Califieri record
... and soon for other records...

B-tree (above) and B+-tree (below) on same data

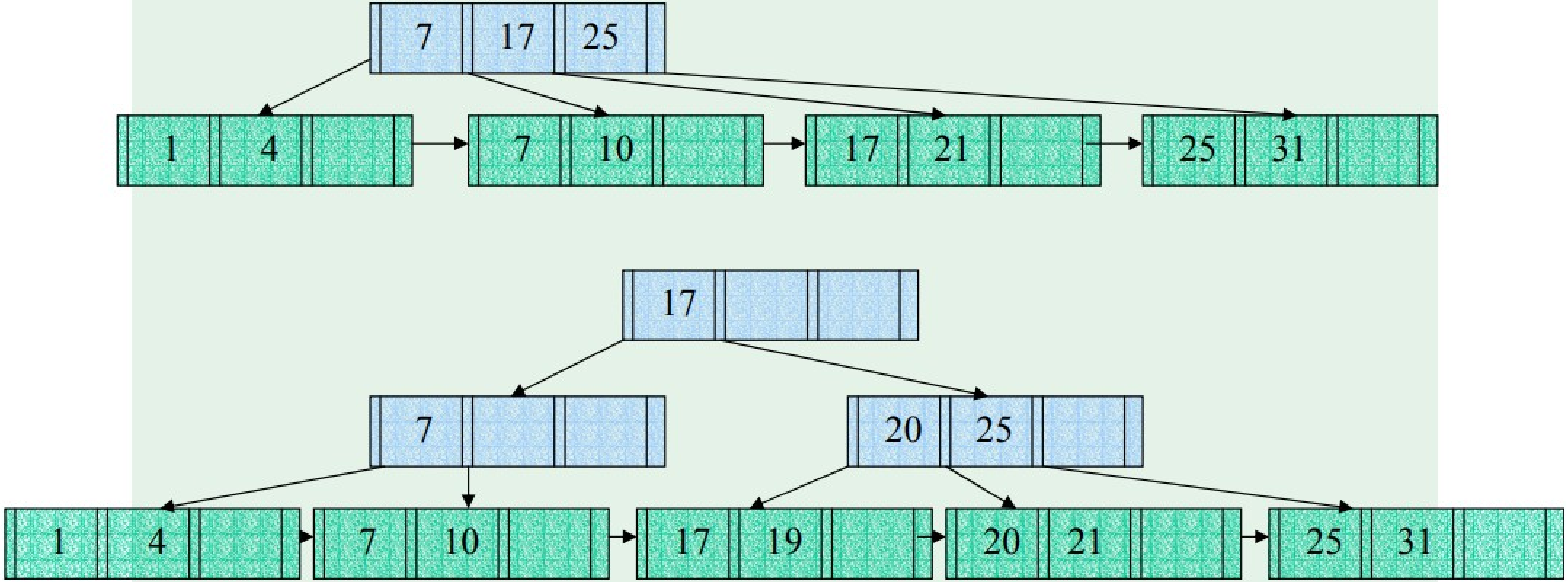

Mozart
Root node
Einstein
Gold
Srinivasan
Internal nodes
Leaf nodes
Brandt
Califieri
Crick
Einstein
El Said
Gold
Katz
Kim
Mozart
Singh
Srinivasan
Wu

Construction : B^{+} -Tree

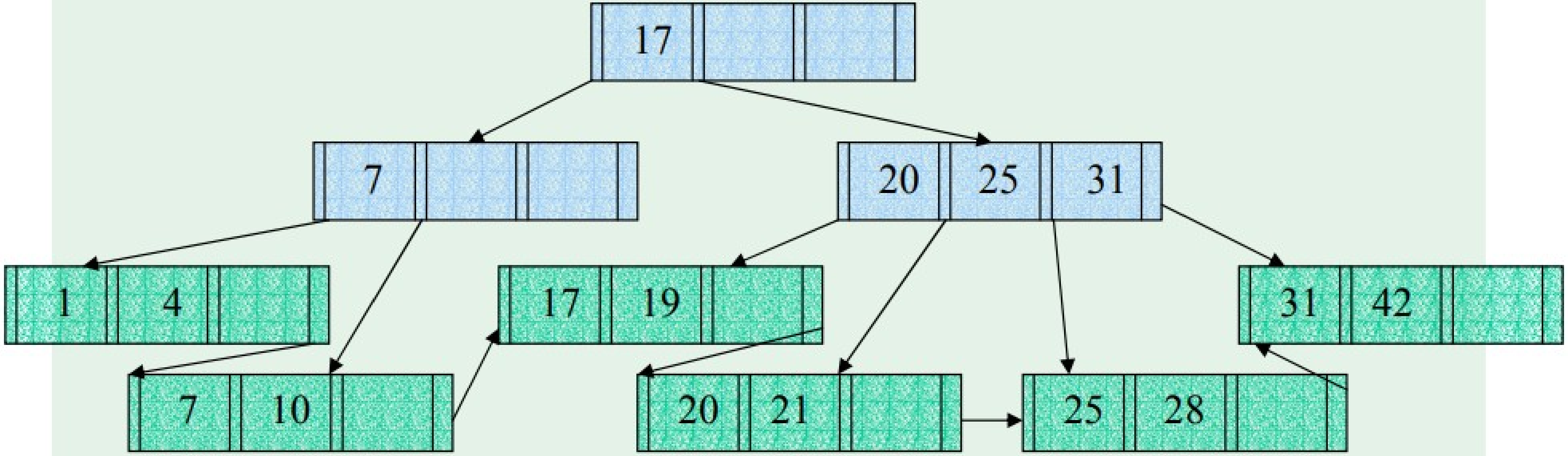
- Example 1: Construct a B^{+} tree for (1, 4, 7, 10, 17, 21, 31, 25, 19, 20, 28, 42) with n=4.


1
4
7
7
1
4
7
10
7
17
1
4
7
10
17
21

- 1, 4, 7, 10, 17, 21, 31, 25, 19, 20, 28, 42

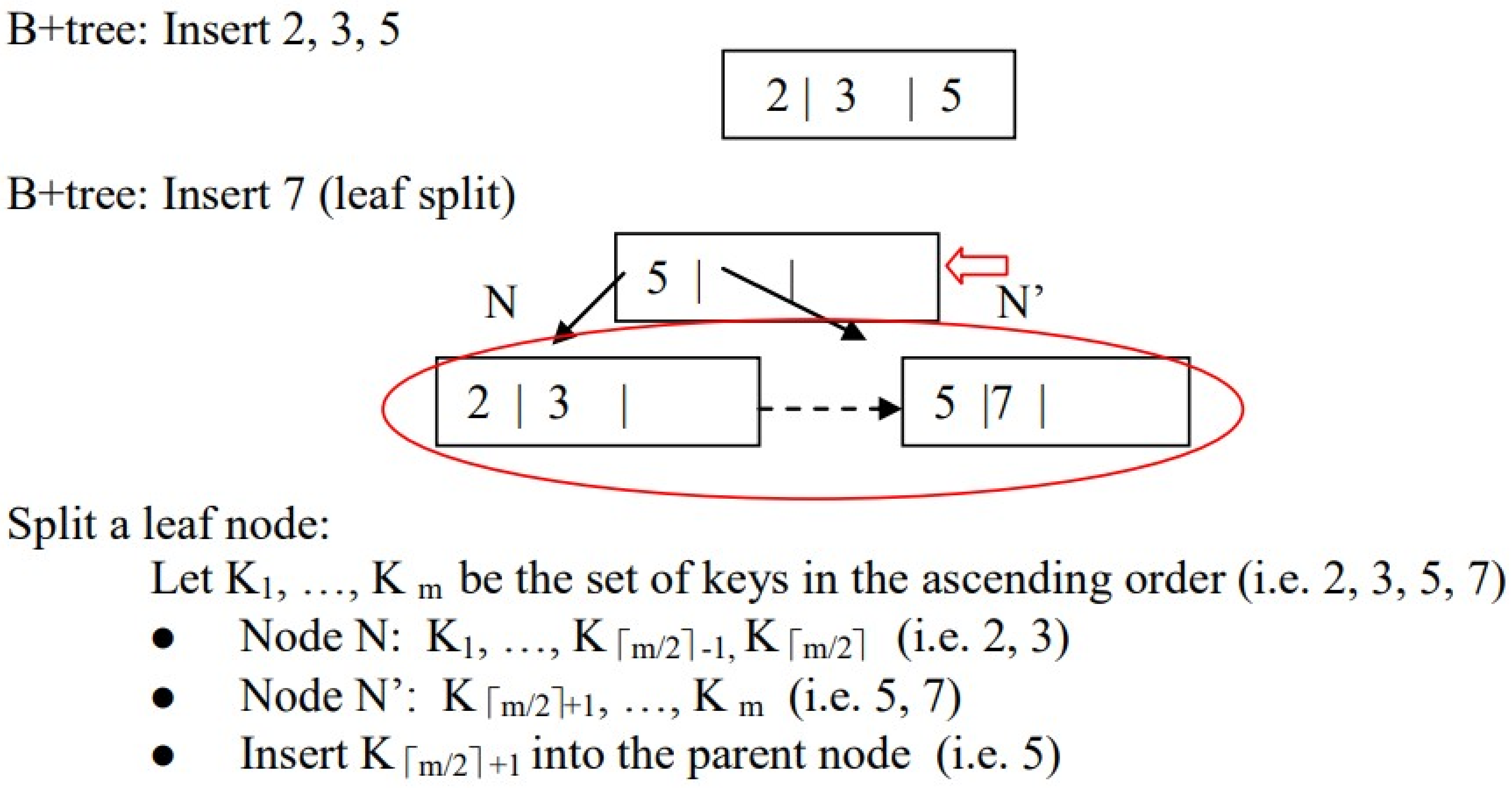

7
17
25
1
4
7
10
17
21
25
31
17
7
20
25
1
4
7
10
17
19
20
21
25
31

- 1, 4, 7, 10, 17, 21, 31, 25, 19, 20, 28, 42


17
7
20
25
31
1
4
17
19
31
42
7
10
20
21
25
28

EXERCISE 1

- Construct a **B⁺** tree for the following set of values (2, 3, 5, 7, 11, 17, 19, 23, 29, 31)
- Assume that the tree is initially empty and values are inserted in ascending order.



B+tree: Insert 2, 3, 5

B+tree: Insert 7 (leaf split)

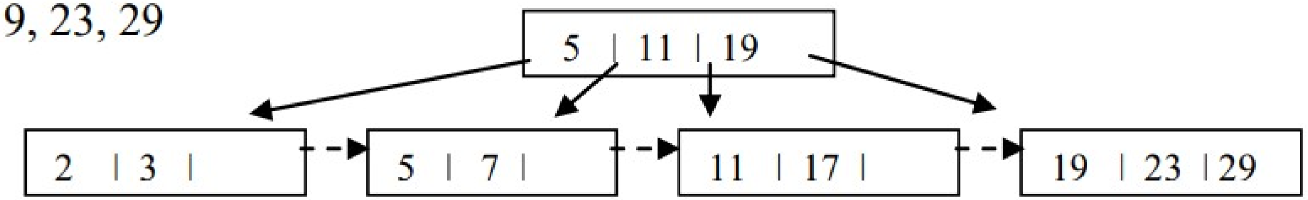
Split a leaf node:

Let $K_1, \ldots, K_m$ be the set of keys in the ascending order (i.e. 2, 3, 5, 7)

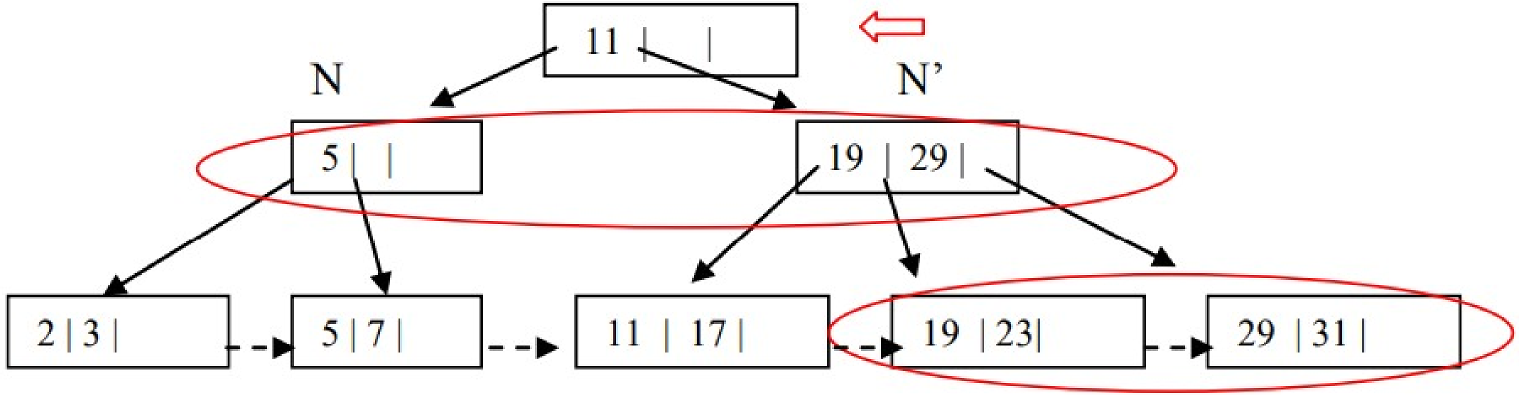
- Node N: $K_1, \ldots, K_{\lceil m/2 \rceil -1}, K_{\lceil m/2 \rceil}$ (i.e. 2, 3)
- Node N': $K_{\lceil m/2 \rceil +1}, \ldots, K_m$ (i.e. 5, 7)
- Insert $K_{\lceil m/2 \rceil +1}$ into the parent node (i.e. 5)

Contd…

B+tree: Insert 11, 17, 19, 23, 29


5 | 11 | 19
2 | 3 |
5 | 7 |
11 | 17 |
19 | 23 | 29

B+tree: Insert 31 (leaf node split + non leaf node split)


11 | |
N
N'
5 | |
19 | 29 |
2 | 3 |
5 | 7 |
11 | 17 |
19 | 23 |
29 | 31 |

Split an internal node N:

Let $K_1, \ldots, K_m$ be the set of keys in the ascending order (i.e. 5, 11, 19, 29)

- Node N: $K_1, \ldots, K_{\lceil m/2 \rceil - 1}$ (i.e. 5)
- Node N': $K_{\lceil m/2 \rceil + 1}, \ldots, K_m$ (i.e. 19, 29)
- Insert $K_{\lceil m/2 \rceil}$ into the parent node of N (i.e. 11)

Web References

- Indexing In DBMS | Why Indexing is used | With Examples | Basics 1/2 – YouTube
- Types Of Indexing In DBMS With Examples | Indexing Basics 2/2 – YouTube
- classification indexing | DBMS – YouTube
- indexing in database example| dbms – YouTube
- B+ tree in database | Introduction & Example | DBMS | Bhanu Priya – YouTube
- https://www.javatpoint.com/b-tree-vs-bplus-tree#:~:text=Let's%20understand%20this%20property%20through,but%20not%20a%20B%20tree

  .

Thank
you

Database Deign

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Contact Session 12:

Transaction Processing, Concurrency Control and Schedules

Transaction Processing, Concurrency Control and Schedules

Objective:

Understanding the Transaction Processing System (TPS), characteristics of the transaction and notions underlying TPS and concurrent transaction execution, Schedules and Serializability.

Contents:

- Introduction to Transactions in DBMS
- Transaction Model
- Concurrent Transactions and Issues
- Understanding Schedules and Serializability
- Concurrency Control and Conflict Serializability
- Test for Conflict Serializability

Introduction to Transactions in DBMS

Types of DBMS

Single-User: at most one user at a time can use the system

Multiuser: many users can use the system concurrently.

Multiprogramming: allows the computer to execute multiple programs at the same time.

Interleaving: keeps the CPU busy when a process requires an input or output operation, the CPU switched to execute another process rather than remaining idle during I/O time.

Most of the theory concerning concurrency control in databases is developed in terms of interleaved concurrency

Introduction to Transactions in DBMS

Execution of transactions in DBMS : Sequential Vs Concurrent Execution

Sequential Execution

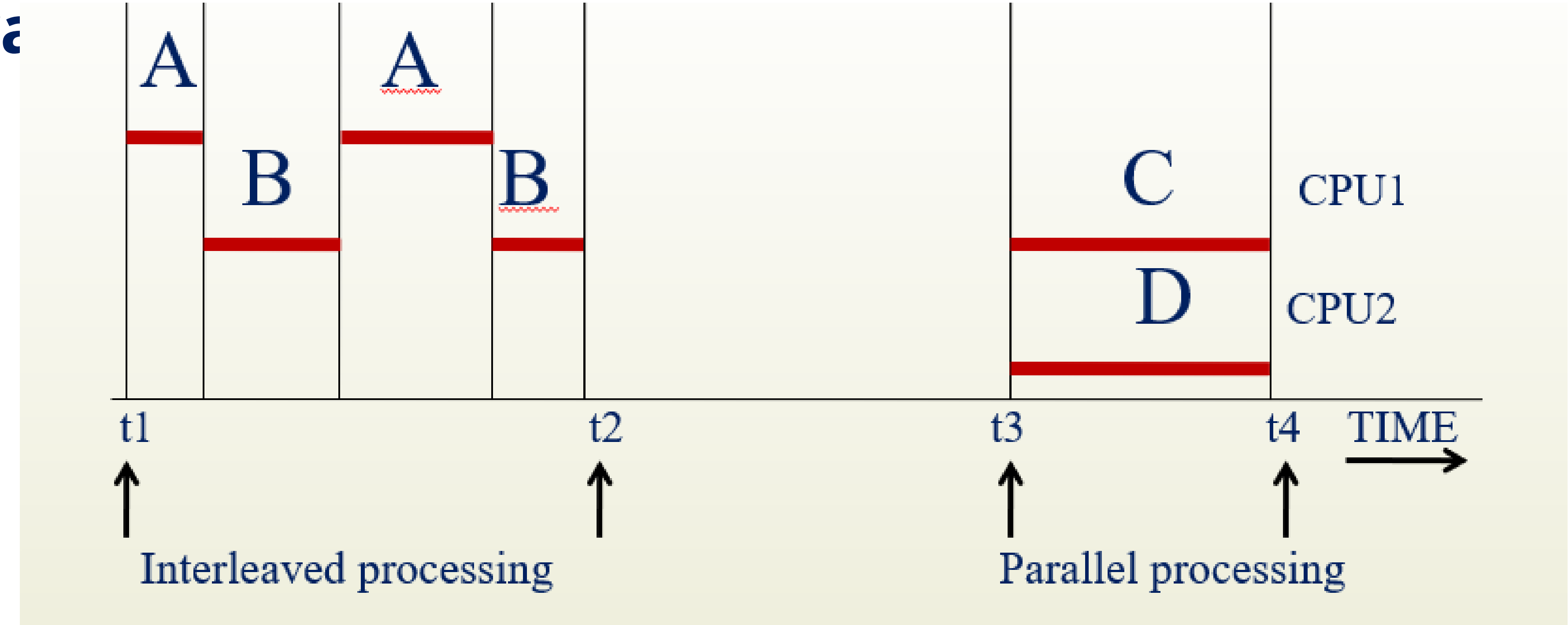
When transactions are strictly executed one after the other it is termed sequential execution

Concurrent Execution

When two or more transactions are interleaved, then we say that it is a concurrent execution

Introduction to Transactions in DBMS

Interleaved Vs parallel processing of the concurrent tra


A
B
A
B
C
D
CPU1
CPU2
t1
t2
t3
t4
TIME
Interleaved processing
Parallel processing

If the computer system has multiple hardware processors (CPUs), **parallel processing** of multiple processes is possible, as illustrated by processes C and D in the above figure.

Introduction to Transactions in DBMS

Transaction

serves as a way to describe logical units of database processing that involve one or more database access operations.

Transaction Processing Systems:

The system with large databases and hundreds of concurrent users that are executing database transactions.

E.g., Banking, Airline reservations, credit card processing, etc.,

Operations on Transaction:

- All database access operations between **Begin Transaction** and **End Transaction** statements are considered one logical transaction.

- If the database operations in a transaction do not update the database but only retrieve data, the transaction is called a **read-only transaction**.

Basic database access operations :

- **read_item(X) :** reads a database item X into program variable.
- **Write_item(X) :** Writes the value of program variable X into the database item X.

Introduction to Transactions in DBMS

- **Example**:

  Suppose an employee of a bank transfers Rs 800 from X's account to Y's account. This small transaction contains several low-level tasks:

X's Account	Y's Account
Open_Account(X)	**Open_Account(Y)**
Old_Balance = X.balance	**Old_Balance = Y.balance**
New_Balance = Old_Balance - 800	**New_Balance = Old_Balance + 800**
X.balance = New_Balance	**Y.balance = New_Balance**
Close_Account(X)	**Close_Account(Y)**

Transaction Model

Transaction

A **transaction** is an atomic unit of work that is either completed in its entirety or not done at all.

- ✓ For recovery purposes, the system needs to keep track of when the transaction starts, terminates, and commits or aborts.
- ✓ **The recovery manager keeps track of the following operations :**
  - **BEGIN_TRANSACTION**
  - **READ OR WRITE**
  - **END_TRANSACTION**
  - **COMMIT_TRANSACTION**
  - **ROLLBACK**

Transaction Model

Transaction states

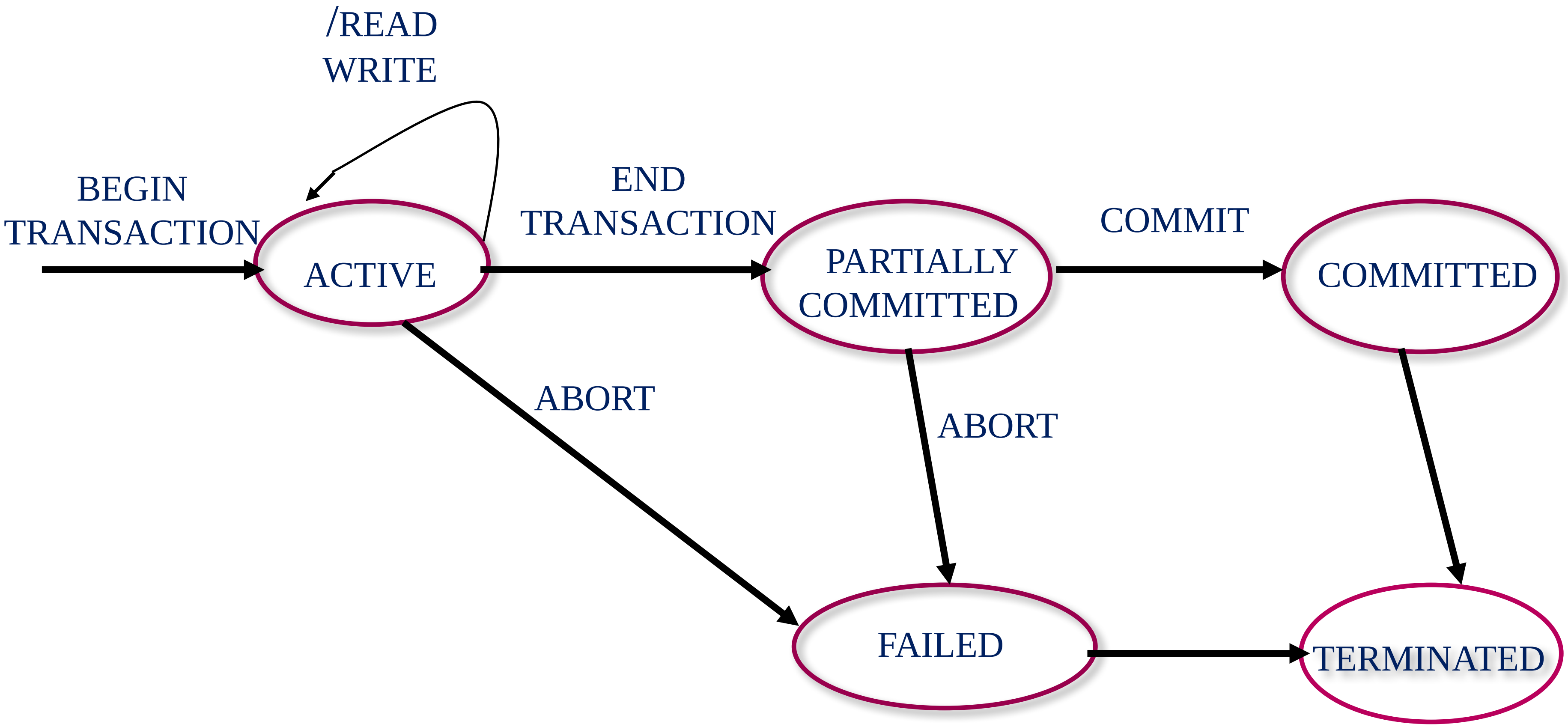

/READ
WRITE
BEGIN
TRANSACTION
ACTIVE
END
TRANSACTION
PARTIALLY
COMMITTED
COMMIT
COMMITTED
ABORT
ABORT
FAILED
TERMINATED

Figure 1: State transition diagram illustrating the states for transaction execution

Transaction Model

The Recovery manager keeps track of the following operations:

Begin Transaction: This is where transaction execution starts. When a transaction begins, it is in its **active state**.

Read/Write: These describe the read and write operations that are performed on database items as part of a transaction.

Commit Transaction: This indicates that the transaction has been completed successfully, allowing any changes (updates) made by the transaction to be safely committed to the database and not undone.

Partially Committed State: When the last statement is executed but the result is not written to the database, this state is achieved.

Failed State: After discovering that the normal execution cannot be continued a transaction is aborted and reaches failed state. In order to ensure atomicity property, failed transactions should have no effect on the database

Transaction Model

Difference between Failed and Partially committed state

If all the 'read and write' operations are performed without any error then it goes to the “partially committed state”; if any instruction fails, it goes to the “failed state”. After completion of all the read and write operation the changes are made in main memory or local buffer.

Rollback/ Abort: This indicates that the transaction failed and that any database modifications or effects caused by the transaction must be undone. As a result, the database must be restored to the state it was in just before the transaction started.

End Transaction: This indicates that read and write transaction activities have finished and that transaction execution has reached its completion.

- It’s reached after the transaction's failure or success
- It may be required at this point to determine if the transaction's changes can be permanently applied to the database, or whether the transaction must be cancelled due to a violation of concurrency control or another

Transaction Model

The Recovery manager keeps track of the following operations…

Undo: It works in a similar way as rollback, however, it only affects a single operation rather than the entire transaction.

Redo:

This specifies that specific transaction activities must be performed in order to

guarantee that a committed transaction's operations have been successfully implemented in the database.

Force writing a log:

- Before a transaction reaches its commit point, any portion of the log that has not been written to the disk yet must now be written to the

Why Recovery is needed?

There are several possible reasons for a transaction to fail:

- ***A computer failure***: A ***hardware, software, or network error occurs*** in the computer system during transaction execution.
- ***A transaction or system error***: Some operations in the transaction may cause it to fail such as ***integer overflow or division by zero***. Transaction failure may also occur because of erroneous parameter values or because of a logical programming error. In addition, ***the user may interrupt the transaction*** during its execution.
- **Local errors or exception conditions**

Detected by the transaction. For example, an ***insufficient account balance*** in a banking database may cause a transaction, such as a fund withdrawal from that account, to be cancelled.

Why Recovery is needed?

- ***Concurrency control enforcement***: The concurrency control method may decide to abort the transaction. Because it ***violates serializability or because several transactions are in a state of deadlock***

- ***Disk failure***: all disks or some disk blocks may lose their data because of a ***disk read/write head crash***

- ***Physical problems***: ***Disasters, theft, fire***, etc.

Therefore, the system must keep sufficient information to recover from the failure.

System Log

The System Log

The system **maintains a log** to keep track of all transaction operations that affect the values of database items. This log may be needed to recover from failures.

Types of log records :

- **[start_transaction,T] :** indicates that transaction T has started execution.
- **[write_item,T,X,old_value,new_value] :** indicates that transaction T has changed the value of database item X from old_value to new_value.

  (new_value may not be recorded)
- **[read_item,T,X]:** indicates that transaction T has read the value of database item X.
- (read_item may not be recorded)
- **[commit,T]:** transaction T has been recorded permanently.
- **[abort,T]:** indicates that transaction T has been aborted.

Transaction Properties

ACID should be enforced by the concurrency control and recovery methods of the DBMS.

Atomicity Consistency Isolation Durability (ACID) properties of transactions :

- **Atomicity:** a transaction is an atomic unit of processing; it is either performed entirely or not performed at all.
- **(It is the responsibility of recovery)**

- **Consistency** : transfer the database from one consistent state to another consistent state [compatible or in agreement with something---standard and effect over time]
- (It is the responsibility of the applications and DBMS to maintain the constraints)

Transaction Properties

- **Isolation**: the execution of the transaction should be isolated from other transactions (Locking) - **(It is the responsibility of concurrency control sub system)**

  * Isolation level:

  -**Level 0 – (No dirty reads):** This means the database will not read any of the uncommitted values

  -Level 1 (No lost update) – A lost update happens when one transaction overwrites the changes made by another transaction without seeing them.

  -Level 2 (No dirty+ no lost)

  -Level 3 (level 2+repeatable reads) –Repeatable reads - This is the most restrictive isolation level. The transaction holds read locks on all rows it references and writes locks on referenced rows for update and delete actions. Since other transactions cannot read, update or delete these rows, consequently it avoids non-repeatable read.

- **Durability**: committed transactions must persist in the database,
- i.e. those changes must not be lost because of any failure.

Concurrent Transactions and Issues

Concurrency Control

In a database management system (DBMS), concurrency control handles simultaneous access to a database.

It prevents two or more users from simultaneously accessing the same record and serializes transactions for backup and recovery.

Why concurrency control is essential?

Several problems can occur when concurrent transactions execute in an uncontrolled manner:

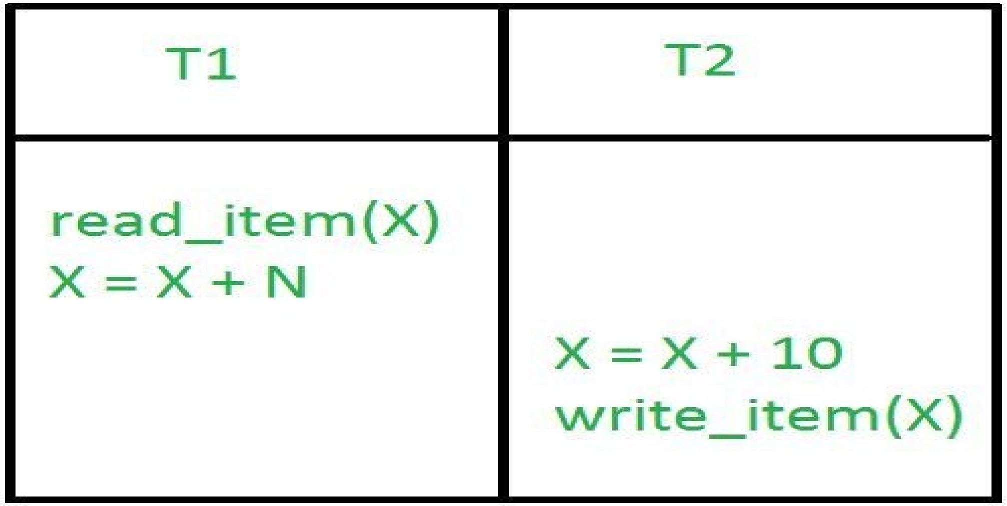
- **The Lost Update Problem**
- **The Temporary Update (or Dirty Read) Problem**
- **The Incorrect Summary Problem**
- **Unrepeatable Read problem**

The Lost update problem

Lost Update Problem:

In the lost update problem, an update done to a data item by a transaction is lost as it is overwritten by the update done by another transaction.

Example 1



T1	T2
read_item(X) X = X + N	
	X = X + 10 write_item(X)

In the above example, transaction 2 changes the value of X but it will get overwritten by the write commit by transaction 1 on X *(not shown in the image above)*. Therefore, the update done by transaction 2 will be lost. Basically, the write commit done by the **last transaction** will overwrite all previous write commits.

Copyright @2007 Ramez Elmasri, Shamkant B. Navathe

Concurrent Transactions and Issues

1. **The Lost Update Problem** : Example 2:

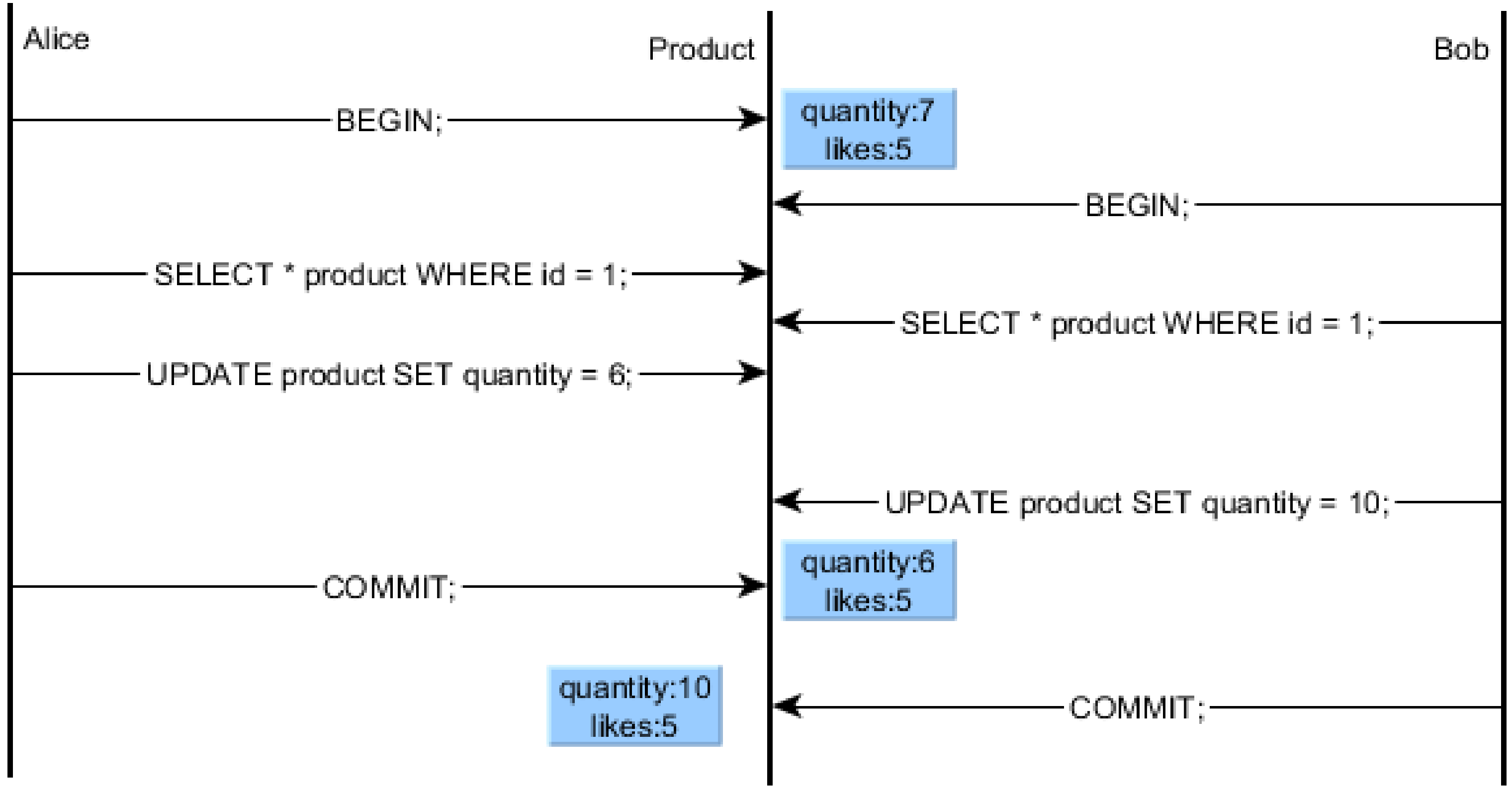
Time

T1	T2
Read_item(X); X=:X-N;	
	Read_item(X); X=:X+M;
Write_item(X); Read_item(Y); Y=:Y+N;	
	Write_item(X);
Write_item(Y);	

Item X has incorrect value

Concurrent Transactions and Issues

Example: 2 **A lost update occurs when two different transactions are trying to update the same column on the same row within a database at the same time**


Alice
Product
Bob
BEGIN;
quantity:7
likes:5
BEGIN;
SELECT * product WHERE id = 1;
SELECT * product WHERE id = 1;
UPDATE product SET quantity = 6;
UPDATE product SET quantity = 10;
COMMIT;
quantity:6
likes:5
quantity:10
likes:5
COMMIT;

Concurrent Transactions and Issues

2. The temporary Update (or Dirty read) problem

Time

T1	T2
Read_item(X); X=:X-N; write_item(X);	
	Read_item(X); X=:X+M; write_item(X);
read_item(Y);	

T1 fails and must rollback, meanwhile T2 has read the temporary value

Concurrent Transactions and Issues

3. The incorrect summary problem

Time

T1	T3
	Sum:=0; Read_item(A); sum:=sum+A; . . .
Read_item(X); X=:X-N; write_item(X);	

	Read_item(X);
	sum:=sum+x;
	read_item(Y);
	sum:=sum+Y;

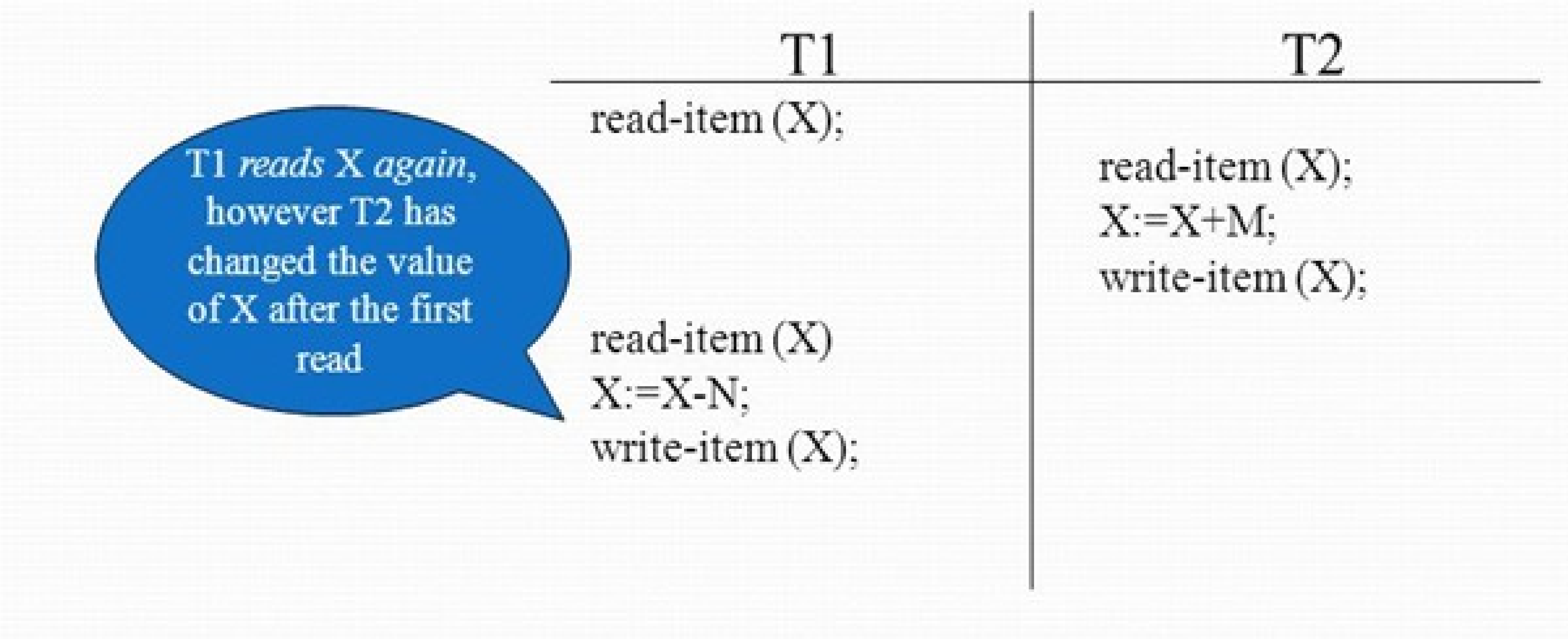
read_item(Y);
Y:=Y+N;
write_item(Y);	

T3 reads X after N is subtracted and reads Y before N is added; A *wrong summary is a result (off by N)*

Concurrent Transactions and Issues

4. **Unrepeatable Read problem**: **A transaction reads items twice with two different values because it was changed by another transaction between the two reads.**

*Also known as **Inconsistent Retrievals Problem that** occurs when in a transaction, two different values are read for the same database item.*



T1	T2
read-item (X);	
	read-item (X);
	X:=X+M;
	write-item (X);
read-item (X)	
X:=X-N;	
write-item (X);	

T1 *reads* X *again*, however T2 has changed the value of X after the first read

Concurrent Transactions and Issues

5. Phantom Read Problem: The phantom read problem happens when a transaction reads a variable once but then encounters an error stating that the variable does not exist when it tries to read it again.

T1	T2
Read (A)	
	Read (A)
Delete (A)	
	Read (A)

Table 1

In the example above Table 1, after transaction 2 reads the variable A, transaction 1 deletes the variable A without the knowledge of transaction 2. As a result, when transaction 2 attempts to read A, it is unable to do so.

Copyright @2007 Ramez Elmasri, Shamkant B. Navathe

Transaction Schedules & Serializability

Schedule(or History) : A series of operations from one transaction to another transaction is known as a schedule. It is used to preserve the order of the operation in each of the individual transaction.

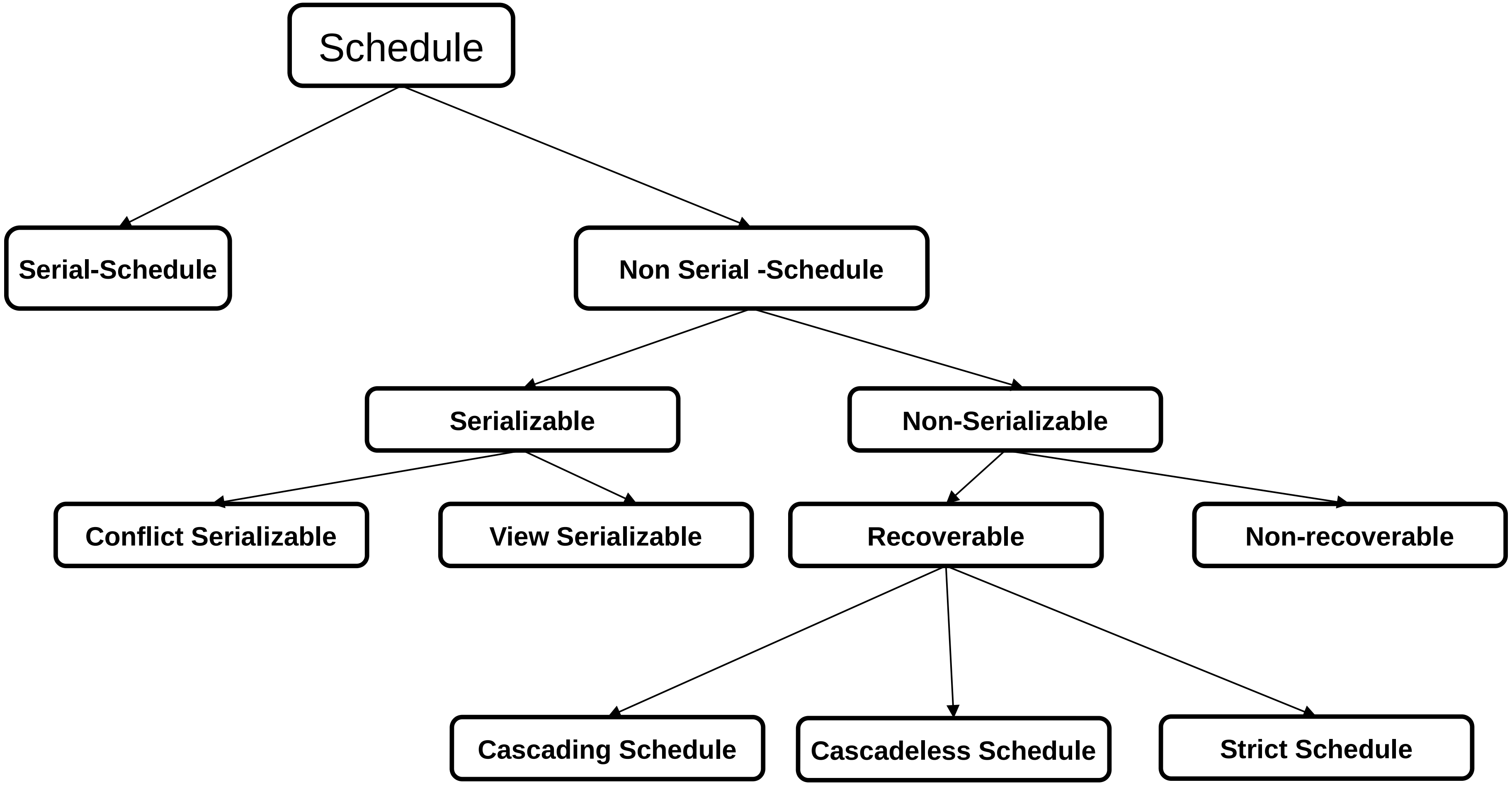
Schedule(or History) S of n transactions T1, T2,.., Tn is an ordering of operations of the transactions with the following conditions :

- for each transaction Ti that participates in S, the operations of Ti in S must appear in the same order in which they occur in Ti
- the operations from other transactions Tj can be interleaved with the operations of Ti in S.

The symbols **r,w,c,** and **a** are used for the operations **read_item, write_item,**

commit, and abort respectively.

Transaction Schedules & Serializability


Schedule
Serial-Schedule
Non Serial -Schedule
Serializable
Non-Serializable
Conflict Serializable
View Serializable
Recoverable
Non-recoverable
Cascading Schedule
Cascadeless Schedule
Strict Schedule

Transaction Schedules & Serializability

Serial Schedule

- In the serial schedule, when the first transaction completes its cycle, then the next transaction is executed.
- All the transactions execute serially one after the other.
- When one transaction executes, no other transaction is allowed to execute.

Characteristics

Serial schedules are always-
- Consistent
- Recoverable
- Cascadeless
- Strict

Time

T1	T2
Read(X);	
X:=X-N	
Write(X);	
Read(Y);	
Y:=Y+N;	
Write(B);	
	Read(X);
	X:=X+M;
	Write(X);

Figure (a)
Schedule A

Schedule A shows the serial schedule where T1 is followed by T2.

Time

T1	T2
	Read(X);
	X:=X+M;
	Write(X);
Read(X);	
X:=X-N	
Write(X);	
Read(Y);	
Y:=Y+N;	
Write(B);	

Figure (b)
Schedule B

Schedule B shows the serial schedu where T2 is followed by T1.

Transaction Schedules & Serializability

Non-serial Schedule

- If interleaving of operations is allowed, then there will be a non-serial schedule.
- It contains many possible orders in which the system can execute the individual operations of the transactions.
- In the given figure (c) and (d), Schedule C and Schedule D are the non-serial schedules. It has interleaving of operations.

Characteristics-

Non-serial schedules are **NOT** always-

- Consistent
- Recoverable
- Cascadeless
- Strict

Transaction Schedules & Serializability

Non-serial Schedule

Time

T1	T2
Read(X); X:=X-N	
	Read(X); X:=X+M;
Write(X); Read(Y);	
	Write(X);
Y:=Y+N; Write(B);	

Figure (c)
Schedule C

Time

T1	T2
Read(X); X:=X-N Write(X);	
	Read(X); X:=X+M; Write(X);
Read(Y); Y:=Y+N; Write(B);	

Figure (d)
Schedule D

Serializable schedule

- The serializability of schedules is used to find non-serial schedules that allow the transaction to execute concurrently without interfering with one another.
- It identifies which schedules are correct when executions of the transaction have to interleave their operations.
- A non-serial schedule will be serializable if its result is equal to the result of its transactions executed serially.

These are of two types:

i) Conflict Serializable

ii) View Serializable

Note: A concurrent schedule whose result is the same as that of a serial schedule is called a ***concurrent serializable schedule***

Transaction Schedules & Serializability

i) **Conflict Serializable :** A schedule S is said to be conflict serializable if it is *conflict equivalent* to some serial schedule S’.

Conflict Equivalent: Two schedules are said to be conflict equivalent when one can be transformed to another by swapping non-conflicting operations.

Conflict Operations:

For transactions T1 & T2 the order of read operation on any data element does not matter.

{T1R(Q), T2R(Q)} or {T2R(Q), T1R(Q)} does not matter. The result is the same and ***does not lead to any conflict***.

Conflicting Operations:

But {T1R(Q), T2W(Q)} is *not* same as {T2W(Q), T1R(Q)}

If **Ii** and **Ij** are the operations (instructions) of two different transactions on the same data item, and at least one of these instructions is a WRITE operation then we say that Ii and Ij ***are conflict operations***.

Here, Q is the data element. **I** stands for instruction and **i** and **j** are transactions.

Transaction Schedules & Serializability

Hence it is evident that if ***we swap non-conflicting operations of a concurrent schedule, it will not affect the final result***.

Example

T_1	T_2
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	
W(B)	
	R(B)
	W(B)

(S_1)

Concurrent schedule with T_1 & T_2 accessing A, B (data item)

T_1	T_2
R(A)	
W(A)	
	R(A)
R(B)	
	W(A)
W(B)	
	R(B)
	W(B)

(S_2)

Swap W(A) in T_2 with R(B) in T_1 (because they are non conflicting)

T_1	T_2
R(A)	
W(A)	
R(B)	
	R(A)
W(B)	
	W(A)
	R(B)
	W(B)

(S_3)

Swap R(A) of T_2 with R(B) of T_1 and W(A) in T_2 with W(B) in T_1 (Since non conflicting)

T_1	T_2
R(A)	
W(A)	
R(B)	
W(B)	
	R(A)
	W(A)
	R(B)
	W(B)

(S_4)

Swap R(A) of T_2 with W(B) of T_1

Now, the final schedule (S_4 is a serial schedule

Transaction Schedules & Serializability

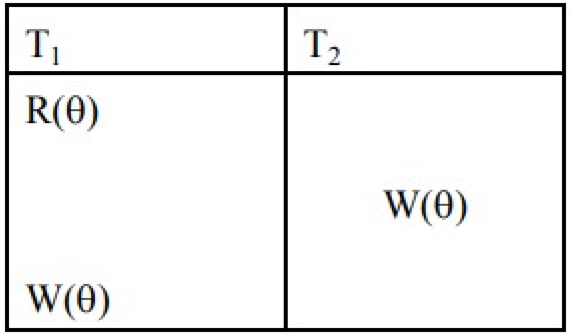
Conflict Equivalent Schedules

We say S and S' are conflict equivalent if a schedule S can be transformed into a schedule S' by swapping non-conflicting instructions.

Furthermore, if a schedule S is ***conflict equivalent*** to a serial schedule, it is said to be ***conflict serializable***.

In the preceding example***, S4 is a serial schedule that is conflict equivalent to S1.***

As a result, S1 is a conflict serializable schedule.



T_1	T_2
R(θ)	
	W(θ)
W(θ)	

Now, in this schedule, we can not perform any swap between instructions of T_1 and T_2
Hence, it is ***not conflict serializable***

Transaction Schedules & Serializability

Test for Conflict Serializability

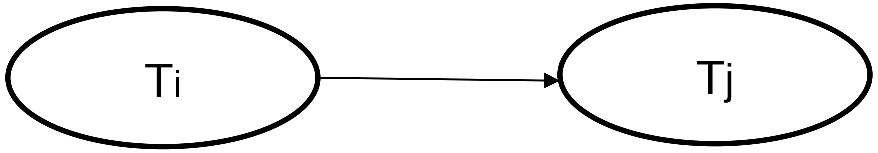
- ✓ Assume a schedule S.
- ✓ Let’s create a graph called a **precedence graph/serialization graph** for S.
- ✓ G = (V, E) is a pair in this graph, where V is a collection of vertices and E is a set of edges.
- ✓ All of the transactions in the schedule are represented by a set of vertices.

- ✓ All edges **Ti ->Tj** that satisfy one of the three conditions are included in the set of edges:

  i. Create a node Ti → Tj if Ti executes write (Q) before Tj executes read (Q).
  ii. Create a node Ti → Tj if Ti executes read (Q) before Tj executes write (Q).
  iii. Create a node Ti → Tj if Ti executes write (Q) before Tj executes write (Q).

Transaction Schedules & Serializability

Precedence Graph for Schedule S:

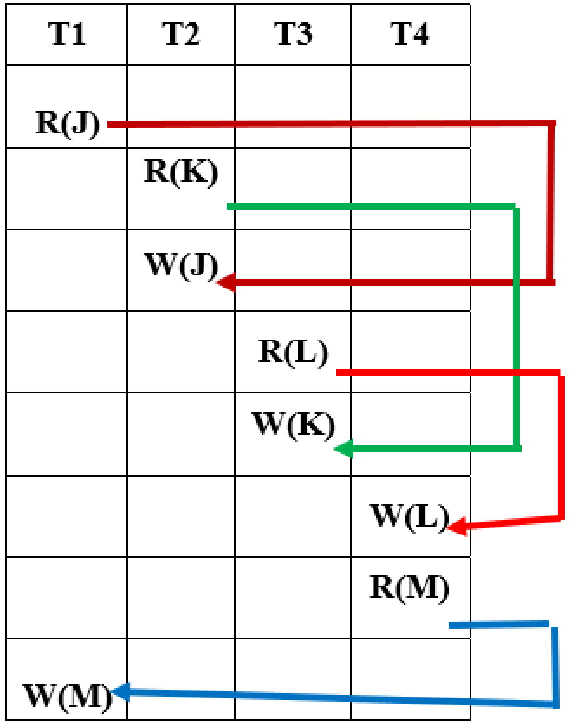


- ✓ If a precedence graph contains a single edge Ti → Tj, then all the instructions of Ti are executed before the first instruction of Tj is executed.
- ✓ If a precedence graph for schedule S contains a cycle, then S is non-serializable. If the precedence graph has no cycle, then S is known as serializable.

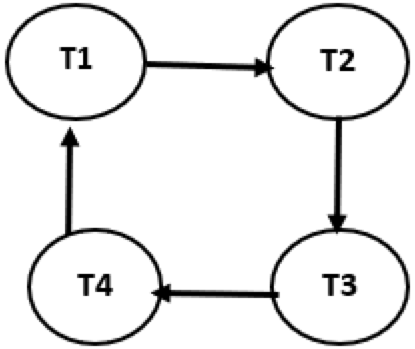
Transaction Schedules & Serializability

Example 1

Sh1: r1(J); r2(K); w2(J); r3(L); w3(K); w4(L);



T1	T2	T3	T4
R(J)			
	R(K)		
	W(J)		
		R(L)	
		W(K)	
			W(L)
			R(M)
W(M)			



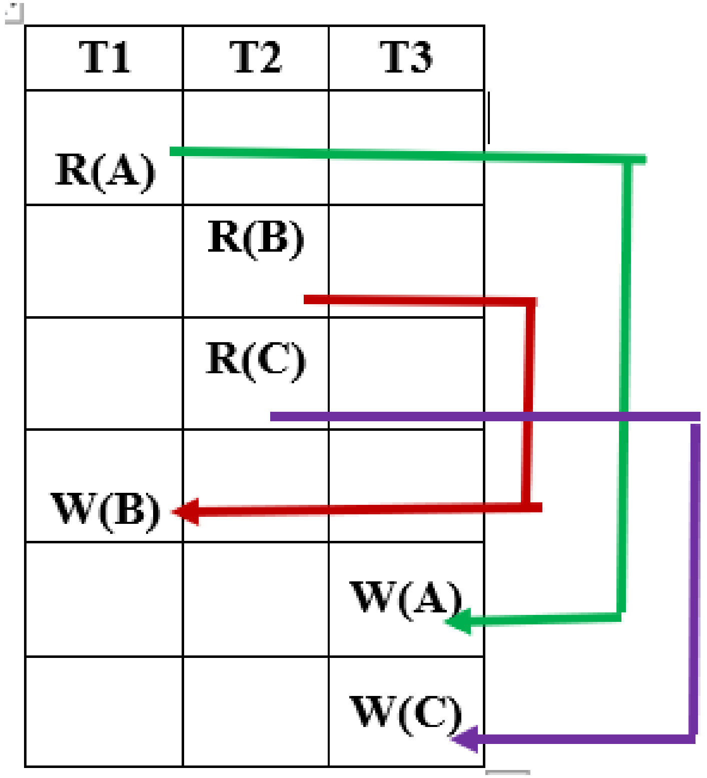
Cycle formed in the precedence graph in the given schedule Sh1.

Therefore, it is "not conflict serializable".

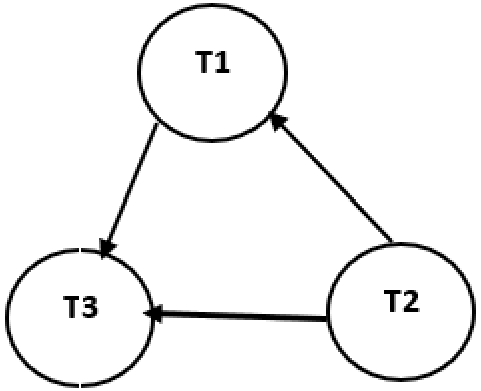
Transaction Schedules & Serializability

Example 2

Sh2: r1(A); r2(B); r2(C); w1(B); w3(A); w3(C);



T1	T2	T3
R(A)		
	R(B)	
	R(C)	
W(B)		
		W(A)
		W(C)



There is no cycle formed in the precedence graph in the given schedule Sh2.

Therefore, it is "Conflict serializable".

Transaction Schedules & Serializability

Conflict equivalent:

- ✓ If the order of any two conflicting operations is the same in both schedules,

  they are said to be conflict equivalent.

- ✓ In the conflict equivalent, one can be transformed into another by swapping

  non- conflicting operations.

- ✓ Two schedules are said to be conflict equivalent if and only if:
  - They contain the same set of transactions.
  - If each pair of conflict operations are ordered in the same way.

Transaction Schedules & Serializability

- ✓ In the following example,

S2 is conflict equivalent to S1

(S1 can be converted to S2 by swapping non-conflicting operations).

T1	T2
Read(X);	
Write(X);	
	Read(X);
	Write(X);
Read(Y);	
Write(Y);	
	Read(Y);
	Write(Y);

Fig 1 Schedule S1

T1	T2
Read(X);	
Write(X);	
Read(Y);	
Write(Y);	
	Read(X);
	Write(X);
	Read(Y);
	Write(Y);

Fig 2 Schedule S2

T1	T2
Read(X);	
Write(X);	
Read(Y);	
Write(Y);	
	Read(X);
	Write(X);
	Read(Y);
	Write(Y);

Fig 3 After swapping of non-conflict operations

- Schedule S2 is a serial schedule because, in this, all operations of T1 are performed before starting any operation of T2.
- Schedule S1 can be transformed into a serial schedule by swapping non-conflicting operations of S1.
- After swapping of non-conflict operations the schedule S1 (fig3), since S1 is conflict serializable).

Schedule S1 becomes

Transaction Schedules & Serializability

Result Equivalent:

When two schedules produce the same final state of the database, they are said to be ***result equivalent.***

Note:

- Being serializable is not the same as being serial
- Being serializable implies that the schedule is the correct schedule.
  - It will leave the database in a consistent state.
  - The interleaving is appropriate and will result in a state as if the transactions were serially executed, yet will achieve efficiency due to concurrent execution.

Transaction Schedules & Serializability

ii) View Serializable

A schedule is view serializable if it is ***view equivalent*** to a serial schedule.

View Equivalent Schedules

Two schedules S and S' (where the same set of transactions participate in both schedules) are said to be view equivalent if the following ***three conditions are met***:

a) For each data item Q, if the transaction Ti reads the initial value of Q in S, then transaction Ti must in schedule S', also read the initial value of Q.

b) For each data item Q, if transaction Ti executes read (Q) in S, and the value produced by transaction Tj (if any) then transaction Ti must in schedule S' also read the value of Q that was produced by transaction Tj

c) For each data item Q, the transaction if any that performs the final write(Q) operation in schedule S must perform the final write(Q) operation in schedule S'

Now, we say that a schedule S is view serializable if it is view equivalent to a serial schedule

Transaction Schedules & Serializability

___Non-Serial____
S1

T1	T2
Read(X);	
Write(X);	
	Read(X);
	Write(X);
Read(Y);	
Write(Y);	
	Read(Y);
	Write(Y);

_____Serial____
S2

T1	T2
Read(X);	
Write(X);	
Read(Y);	
Write(Y);	
	Read(X);
	Write(X);
	Read(Y);
	Write(Y);

We can say that given schedule s1 is view serializable if we can prove that they are *view equivalent*.

Note: Every conflict serializable schedule is view serializable. But not all view serializable schedules are conflict serializable.

Transaction Schedules & Serializability

Non-Serializability in DBMS

A non-serial schedule, which is not serializable is called as non-serializable schedule.

Non-serializable schedules may or may not be consistent or recoverable.

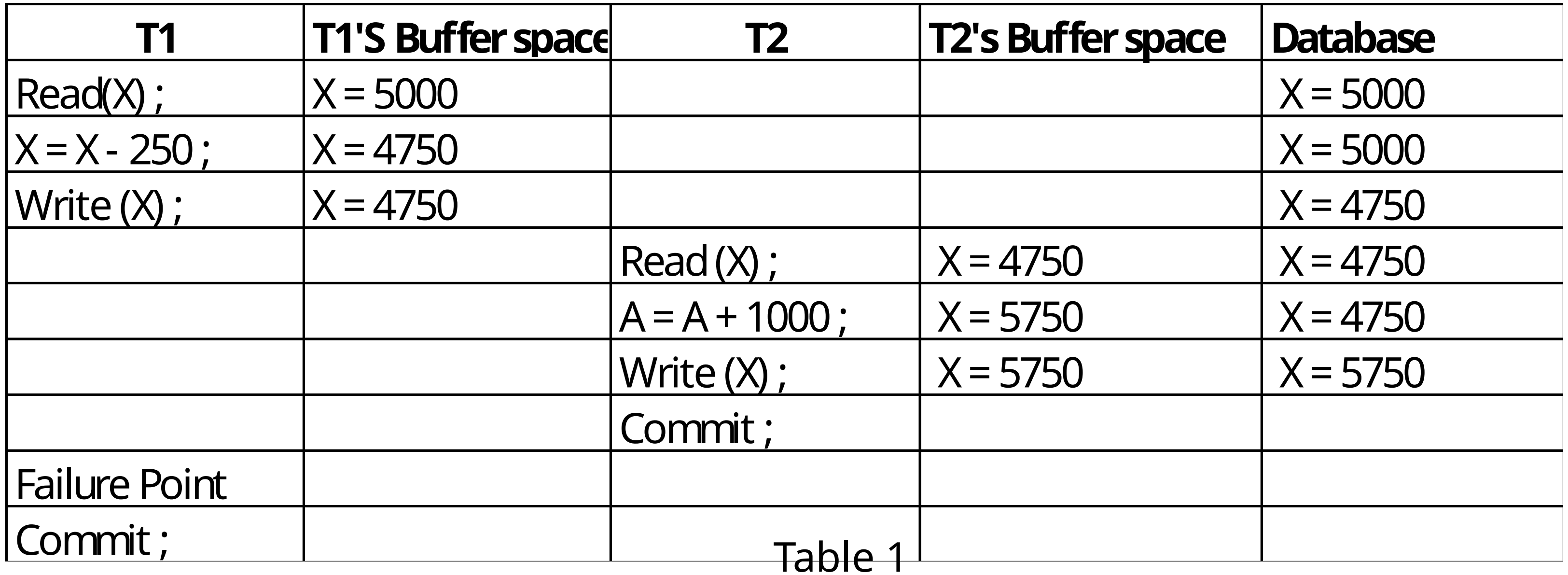
Non-serializable schedule is divided into types:

i. **Recoverable schedule**
ii. **Non-recoverable schedule**

Recoverable Schedule

Due to a software issue, system crash, or hardware malfunction, a transaction may not finish completely. The unsuccessful transaction must be rolled back in this scenario. However, the value generated by the unsuccessful transaction may have been utilized by another transaction. As a result, we must also roll back those transactions.

Non-recoverable/ Irrecoverable Schedule



T1	T1'S Buffer space	T2	T2's Buffer space	Database
Read(X) ;	X = 5000			X = 5000
X = X - 250 ;	X = 4750			X = 5000
Write (X) ;	X = 4750			X = 4750
		Read (X) ;	X = 4750	X = 4750
		A = A + 1000 ;	X = 5750	X = 4750
		Write (X) ;	X = 5750	X = 5750
		Commit ;		
Failure Point				
Commit ;				

Table 1

The above Table 1 shows a schedule that has two transactions.

T1 reads and writes the value of A and that value is read and written by T2.

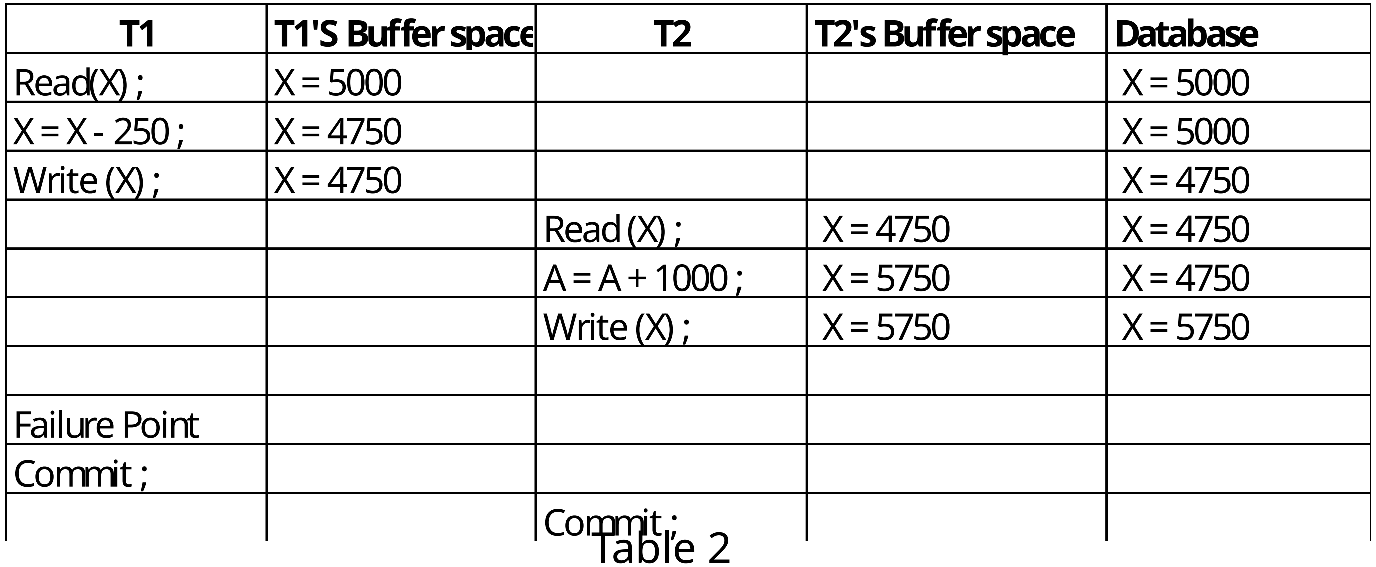
T2 commits but later on, T1 fails. Due to the failure, we have to rollback T1.

T2 should also be rollback because it reads the value written by T1, but T2 can't be rollback because it is already committed. So this type of schedule is known as ***“non-recoverable/irrecoverable”*** schedule.

Irrecoverable schedule: The schedule will be irrecoverable if Tj reads the updated value of Ti and Tj committed before Ti commit.

Transaction Schedules & Serializability

Recoverable with Cascade Rollback



T1	T1'S Buffer space	T2	T2's Buffer space	Database
Read(X) ;	X = 5000			X = 5000
X = X - 250 ;	X = 4750			X = 5000
Write (X) ;	X = 4750			X = 4750
		Read (X) ;	X = 4750	X = 4750
		A = A + 1000 ;	X = 5750	X = 4750
		Write (X) ;	X = 5750	X = 5750
Failure Point				
Commit ;				
		Commit ;		

Table 2

The above Table 2 shows a schedule with two transactions.

Transaction T1 reads and writes A, and that value is read and written by transaction T2. But later on, T1 fails. Due to this, we have to rollback T1.

T2 should be rollback because T2 has read the value written by T1. As it has not committed before T1 commits so we can rollback transaction T2 as well. So it is ***recoverable with cascade rollback***.

Recoverable with cascading rollback: The schedule will be recoverable with cascading rollback if Tj reads the updated value of Ti. Commit of Tj is delayed till commit of Ti.

Transaction Schedules & Serializability

Cascadeless Recoverable Schedule

T1	T1'S Buffer space	T2	T2's Buffer space	Database
Read(X) ;	X = 5000			X = 5000
X = X - 250 ;	X = 4750			X = 5000
Write (X) ;	X = 4750			X = 4750
Commit ;		Read (X) ;	X = 4750	X = 4750
		A = A + 1000 ;	X = 5750	X = 4750
		Write (X) ;	X = 5750	X = 5750
		Commit ;		

Table 3

The above Table 3 shows a schedule with two transactions.

Transaction T1 reads and write A and commits, and that value is read and written by T2.

So this is a ***cascade less recoverable schedule***.

Transaction Schedules & Serializability

Strict Recoverable Schedule

When a transaction is not allowed to access or write data until the last transaction that has written it is committed or aborted, the schedule is referred to as Strict Schedule.

T1	T2
Read(X) ;	
	Read(X) ;
Write (X) ;	
Commit ;	
	Write (X) ;
	Read (X) ;
	Commit ;

Table 4

Here, transaction T2 reads/writes the written value of transaction T1 only after the transaction T1 commits. Hence, the schedule is a ***strict schedule.***

Let's have two transactions T1 and T2 in Table 4.

The write operation of transaction T1 precedes the read or write operation of transaction T2, so the commit or abort operation of transaction T1 should also precede the read or write of T2.

The Strict schedule allows only committed read and write operations.

This schedule implements more restrictions than cascadeless schedule.

SUMMARY

- ✓ **Transactions in DBMS**
- ✓ **Concurrent Transactions and Issues**
- ✓ **Schedules in DBMS**
  - Types of schedules
    - ❖ **Serial Schedule**
    - ❖ **Non-serial Schedule**
      - ➢ **Serializable**
        - Conflict Serializable & Test for Conflict Serializability
        - View Serializable
      - ➢ **Non-serializable**
        - **- Non-Recoverable Schedule**
        - **- Recoverable Schedule**
          - Cascading Schedule
          - Cascadeless Schedule
          - Strict Schedule

Thank
you

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

Database Design

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

Contact Session 13:

Concurrency Control, Lock-based Protocols and Deadlocks, Database Recovery

Contents:

- Concurrency control and Implementing Serializability
- Lock-based protocols and Deadlocks
- Timestamp based protocols
- Introduction to Database recovery and Log-based recovery
- Checkpointing
- Shadow paging

Let's have a quick recap!!

Transaction

- Both concurrency control and crash recovery are based on the concept of a **transaction**.
- A transaction is a set of SQL statements chosen by the user.
- Example:

  Transfer $100 from

  one account to another:

```
BEGIN transaction
read balance from first account
add $100 to first account balance
write balance to first account
read balance from second account
verify balance to see if it contains at least $100
        if not, ABORT transaction
subtract $100 from second account
write balance to second account
COMMIT transaction
```

The ACID Properties of Transactions

- **Atomicity**: A transaction happens in its entirety or not at all
  - What if the OS crashed after $100 was deposited to the first account?
  - The *recovery manager* of the DBMS must assure that the $100 is withdrawn from the first account.
- **Consistency:** If the DB starts in a consistent state, (this notion is defined by the user; some of it may be enforced by integrity constraints) the transaction will transform it into a consistent state.
  - What if a transaction just deposited $100 into an account?
  - The *programmer* must ensure that all transactions are consistent.
- **Isolation:** Each transaction is isolated from other transactions. The effect on the DB is as if the transaction executed by itself.
  - What if another transaction computed the total bank balance after $100 was deposited to the first account?
  - The concurrency control subsystem must ensure that all transactions run in isolation, unless the DBA chooses a less strict level of isolation.
- **Durability:** If a transaction commits, its changes to the database state persist
  - What if, after the commit, the OS crashed before the withdrawal was written to disk?
  - The recovery manager must assure that the withdrawal was at least logged.

Concurrency

- First we will study isolation of transactions, ensured by the concurrency control subsystem.
- Isolation is a problem only when multiple users are accessing the same data, and their actions interleave.
- Why is concurrency necessary?
  - Applications demand it
  - Better utilization of resources: While one user/transaction is reading the disk, another can be using the CPU or reading another disk. This results in better throughput and response time.

Let’s start today’s session!!

Purpose of Concurrency Control

- To enforce Isolation (through mutual exclusion) among conflicting transactions.
- To preserve database consistency through consistency preserving execution of transactions.
- To resolve read-write and write-write conflicts.

Example:

- In concurrent execution environment if T1 conflicts with T2 over a data item A, then the existing concurrency control decides if T1 or T2 should get the A and if the other transaction is rolled-back or waits.

Two-Phase Locking Techniques

- Locking is an operation which secures
  - (a) permission to Read
  - (b) permission to Write a data item for a transaction.
- Example:
  - Lock (X). Data item X is locked on behalf of the requesting transaction.
- Unlocking is an operation which removes these permissions from the data item.
- Example:
  - Unlock (X): Data item X is made available to all other transactions.
- Lock and Unlock are Atomic operations.

Two-Phase Locking Techniques: Essential components

- Two locks modes:
  - (a) shared (read) (b) exclusive (write).
- Shared mode: shared lock (X)
  - More than one transaction can apply share lock on X for reading its value but no write lock can be applied on X by any other transaction.
- Exclusive mode: Write lock (X)
  - Only one write lock on X can exist at any time and no shared lock can be applied by any other transaction on X.
- Conflict matrix

	Read	Write
Read	Y	N
Write	N	N

Two-Phase Locking Techniques: Essential components

- Lock Manager:
  - Managing locks on data items.
- Lock table:
  - Lock manager uses it to store the identity of transaction locking a data item, the data item, lock mode and pointer to the next data item locked. One simple way to implement a lock table is through linked list.

Transaction ID	Data item id	lock mode	Ptr to next data item
T1	X1	Read	Next

Two-Phase Locking Techniques: Essential components

- Database requires that all transactions should be well-formed. A transaction is well-formed if:
    - It must lock the data item before it reads or writes to it.
    - It must not lock an already locked data items and it must not try to unlock a free data item.

Two-Phase Locking Techniques: Essential components

- The following code performs the lock operation:

```
B: if LOCK (X) = 0 (*item is unlocked*)
   then LOCK (X) ← 1 (*lock the item*)
   else begin
            wait (until lock (X) = 0) (until
            the lock manager wakes up the transaction);
   goto B
   end;
```

Two-Phase Locking Techniques: Essential components

- The following code performs the unlock operation:

```
LOCK (X) ← 0 (*unlock the item*)
if any transactions are waiting then
        wake up one of the waiting transactions;
```

Two-Phase Locking Techniques: Essential components

- The following code performs the read operation:

```
B: if LOCK (X) = “unlocked” then
  begin LOCK (X) ← “read-locked”;
      no_of_reads (X) ← 1;
  end
  else if LOCK (X) ← “read-locked” then
          no_of_reads (X) ← no_of_reads (X) +1
       else begin wait (until LOCK (X) = “unlocked” and
                    the lock manager wakes up the transaction);
                    go to B
                end;
```

Two-Phase Locking Techniques: Essential components

- The following code performs the write lock operation:

```
B: if LOCK (X) = “unlocked” then
 begin LOCK (X) ← “write-locked”;
     no_of_writes (X) ← 1;
 end
 else if LOCK (X) ← “read-locked” then
           begin wait (until LOCK (X) = “unlocked” and
                   the lock manager wakes up the transaction);
                   go to B
                end;
```

Two-Phase Locking Techniques: Essential components

- The following code performs the unlock operation:

```
if LOCK (X) = “write-locked” then
  begin LOCK (X) ← “unlocked”;
        wakes up one of the transactions, if any
  end
  else if LOCK (X) ← “read-locked” then
       begin
             no_of_reads (X) ← no_of_reads (X) -1
             if  no_of_reads (X) = 0 then
             begin
                       LOCK (X) = “unlocked”;
                      wake up one of the transactions, if any
             end
       end;
```

Two-Phase Locking Techniques: Essential components

- Lock conversion
    - Lock upgrade: existing read lock to write lock

```
        if Ti has a read-lock (X) and Tj has no read-lock (X) (i ≠ j) then
    convert read-lock (X) to write-lock (X)
            else
    force Ti to wait until Tj unlocks X
```

- Lock downgrade: existing write lock to read lock

```
        Ti has a write-lock (X)    (*no transaction can have any lock on X*)
        convert write-lock (X) to read-lock (X)
```

Two-Phase Locking Techniques: The algorithm

- Two Phases:
  - (a) Locking (Growing)
  - (b) Unlocking (Shrinking).
- **Locking (Growing) Phase:**
  - A transaction applies locks (read or write) on desired data items one at a time.
- **Unlocking (Shrinking) Phase:**
  - A transaction unlocks its locked data items one at a time.
- **Requirement:**
  - For a transaction these two phases must be **mutually exclusive**, that is, during locking phase unlocking phase must not start and during unlocking phase locking phase must not begin.

Two-Phase Locking Techniques: The algorithm

T1	**T2**	**Result**
read_lock (Y);	read_lock (X);	Initial values: X=20; Y=30
read_item (Y);	read_item (X);	Result of serial execution
unlock (Y);	unlock (X);	T1 followed by T2
write_lock (X);	Write_lock (Y);	X=50, Y=80.
read_item (X);	read_item (Y);	Result of serial execution
X:=X+Y;	Y:=X+Y;	T2 followed by T1
write_item (X);	write_item (Y);	X=80, Y=50
unlock (X);	unlock (Y);	

Two-Phase Locking Techniques: The algorithm

T1	**T2**
read_lock (Y); read_item (Y); **unlock (Y);** (WHY?)	
	read_lock (X); read_item (X); **unlock (X);** **write_lock (Y);** read_item (Y); Y:=X+Y; write_item (Y); unlock (Y);
write_lock (X); read_item (X); X:=X+Y; write_item (X); unlock (X);	

Time

Result

X=50; Y=50
Non-serializable because it.
violated two-phase policy.

Conditions for Two-phase locking protocol

- The two-phase locking divides the transaction execution phase into three components.
- Phase 1: When the transaction's execution begins in the first part, it requests permission for the lock it requires.
- Phase 2The transaction then obtains all of the locks in the second part.
- Phase 3: The third phase begins when the transaction's first lock is released.
- The transaction cannot demand any new locks in the third phase. It only unlocks the locks that have been acquired.

Conditions for Two-phase locking protocol

- A transaction follows the two-phase locking protocol if locking and unlocking can be done in two phases. The two phases of the two-phase protocol are:
- **Growing phase:** During the growth phase, new locks on data items may be acquired, but none can be released.
- **Shrinking phase:** Existing locks may be released, but no new locks can be acquired during the shrinking phase.
- **Note:** If lock conversion is allowed, the following phase may take place:
- Upgrading of lock (from S(a) → X(a)) is allowed in the growing phase.
- Downgrading of lock (from X(a) → S(a)) must be done in the shrinking phase.
- **Lock Point:** It is a point at which the growing phase comes to a close, i.e., when a transaction obtains the last lock it requires.
- Let's look at an example:

Conditions for Two-phase locking protocol

	T1	T2
1	Lock - S(A)	
2		Lock - S(A)
3	Lock - X(B)	
4	…….	…….
5	Unlock(A)	
6		Lock - X(C)
7	Unlock(B)	
8		Unlock(A)
9		Unlock(C)
10	…….	…….

The way unlocking and locking works in two-phase locking is:

Transaction T1:

- The growing phase is from steps 1-3.
- The shrinking phase is from steps 5-7.
- The locking point is at point 3.

Transaction T2:

- The growing phase is from steps 2-6.
- The shrinking phase is from steps 8-9.
- The locking point is at point 6.

The type mentioned above of 2-PL is Basic 2PL. It ensures Serializability, but it does not prevent Cascading Rollback and Deadlock.

Two-Phase Locking Techniques: The algorithm

T’1	**T’2**
read_lock (Y);	read_lock (X);
read_item (Y);	read_item (X);
write_lock (X);	Write_lock (Y);
unlock (Y);	unlock (X);
read_item (X);	read_item (Y);
X:=X+Y;	Y:=X+Y;
write_item (X);	write_item (Y);
unlock (X);	unlock (Y);

T1 and T2 follow two-phase policy but they are subject to deadlock, which must be dealt with.

Two-Phase Locking Techniques: The algorithm

- Two-phase policy generates two locking algorithms
  - (a) **Basic**
  - (b) **Conservative**
- **Conservative**:
  - Prevents deadlock by locking all desired data items before transaction begins execution.
- **Basic**:
  - Transaction locks data items incrementally. This may cause deadlock which is dealt with.
- **Strict**:tp
- A more stricter version of Basic algorithm where unlocking is performed after a transaction terminates (commits or aborts and rolled-back).
- This is the most commonly used two-phase locking algorithm.

Dealing with Deadlock and Starvation

- **Deadlock**

T’1	**T’2**
read_lock (Y);	
read_item (Y);	
	read_lock (X);
	read_item (Y);
write_lock (X);	
(waits for X)	write_lock (Y);
	(waits for Y)

T1 and T2 did follow two-phase policy but they are deadlock

- Deadlock (T’1 and T’2)

Dealing with Deadlock and Starvation

- **Deadlock prevention**
  - A transaction locks all data items it refers to before it begins execution.
  - This way of locking prevents deadlock since a transaction never waits for a data item.
  - The conservative two-phase locking uses this approach.

Dealing with Deadlock and Starvation

- **Deadlock detection and resolution**
  - In this approach, deadlocks are allowed to happen.
  - The scheduler maintains a wait-for-graph for detecting cycle.
  - If a cycle exists, then one transaction involved in the cycle is selected (victim) and rolled-back.
  - A wait-for-graph is created using the lock table.
  - As soon as a transaction is blocked, it is added to the graph.

  - When a chain like: Ti waits for Tj waits for Tk waits for Ti or Tj occurs, then this creates a cycle.
  - One of the transaction should be rolled back.

Dealing with Deadlock and Starvation

- **Deadlock avoidance**
  - There are many variations of two-phase locking algorithm.
  - Some avoid deadlock by not letting the cycle to complete.
  - That is as soon as the algorithm discovers that blocking a transaction is likely to create a cycle, it rolls back the transaction.
  - Wound-Wait and Wait-Die algorithms use timestamps to avoid deadlocks by rolling-back victim.

Dealing with Deadlock and Starvation

- **Starvation**
    - Starvation occurs when a particular transaction consistently waits or restarted and never gets a chance to proceed further.
    - In a deadlock resolution it is possible that the same transaction may consistently be selected as victim and rolled-back.
    - This limitation is inherent in all priority based scheduling mechanisms.
    - In Wound-Wait scheme a younger transaction may always be wounded (aborted) by a long running older transaction which may create starvation.

Timestamp based concurrency control algorithm

- **Timestamp**
  - A monotonically increasing variable (integer) indicating the age of an operation or a transaction.
  - A larger timestamp value indicates a more recent event or operation.
  - Timestamp based algorithm uses timestamp to serialize the execution of concurrent transactions.

Timestamp based concurrency control algorithm

- **Basic Timestamp Ordering**
  - 1. Transaction T issues a write_item(X) operation:
    - If read_TS(X) > TS(T) or if write_TS(X) > TS(T), then an younger transaction has already read the data item so abort and roll-back T and reject the operation.
    - If the condition in part (a) does not exist, then execute write_item(X) of T and set write_TS(X) to TS(T).
  - 2. Transaction T issues a read_item(X) operation:
    - If write_TS(X) > TS(T), then an younger transaction has already written to the data item so abort and roll-back T and reject the operation.
    - If write_TS(X) ≤ TS(T), then execute read_item(X) of T and set read_TS(X) to the larger of TS(T) and the current read_TS(X).

Timestamp based concurrency control algorithm

- **Strict Timestamp Ordering**
  - 1. Transaction T issues a write_item(X) operation:
    - If TS(T) > read_TS(X), then delay T until the transaction T’ that wrote or read X has terminated (committed or aborted).
  - 2. Transaction T issues a read_item(X) operation:
    - If TS(T) > write_TS(X), then delay T until the transaction T’ that wrote or read X has terminated (committed or aborted).

Timestamp based concurrency control algorithm

- **Thomas's Write Rule**
  - If read_TS(X) > TS(T) then abort and roll-back T and reject the operation.
  - If write_TS(X) > TS(T), then just ignore the write operation and continue execution. This is because the most recent writes counts in case of two consecutive writes.
  - If the conditions given in 1 and 2 above do not occur, then execute write_item(X) of T and set write_TS(X) to TS(T).

Databases Recovery

Purpose of Database Recovery

- To bring the database into the last consistent state, which existed prior to the failure.
- To preserve transaction properties (Atomicity, Consistency, Isolation and Durability).

Example:

- If the system crashes before a fund transfer transaction completes its execution, then either one or both accounts may have incorrect value.
- Thus, the database must be restored to the state before the transaction modified any of the accounts.

Types of Failure

- The database may become unavailable for use due to
    - **Transaction failure**: Transactions may fail because of incorrect input, deadlock, incorrect synchronization.
    - **System failure**: System may fail because of addressing error, application error, operating system fault, RAM failure, etc.
    - **Media failure**: Disk head crash, power disruption, etc.

Transaction Log

- For recovery from any type of failure data values prior to modification (BFIM - BeFore Image) and the new value after modification (AFIM – AFter Image) are required.
- These values and other information is stored in a sequential file called Transaction log. A sample log is given below. Back P and Next P point to the previous and next log records of the same transaction.

T ID	Back P	Next P	Operation	Data item	BFIM	AFIM
T1	0	1	Begin			
T1	1	4	Write	X	X = 100	X = 200
T2	0	8	Begin			
T1	2	5	W	Y	Y = 50	Y = 100
T1	4	7	R	M	M = 200	M = 200
T3	0	9	R	N	N = 400	N = 400
T1	5	nil	End			

Database recovery techniques

Data Update

- **Immediate Update**: As soon as a data item is modified in cache, the disk copy is updated.
- **Deferred Update**: All modified data items in the cache is written either after a transaction ends its execution or after a fixed number of transactions have completed their execution.
- **Shadow update**: The modified version of a data item does not overwrite its disk copy but is written at a separate disk location.
- **In-place update**: The disk version of the data item is overwritten by the cache version.

Data Caching

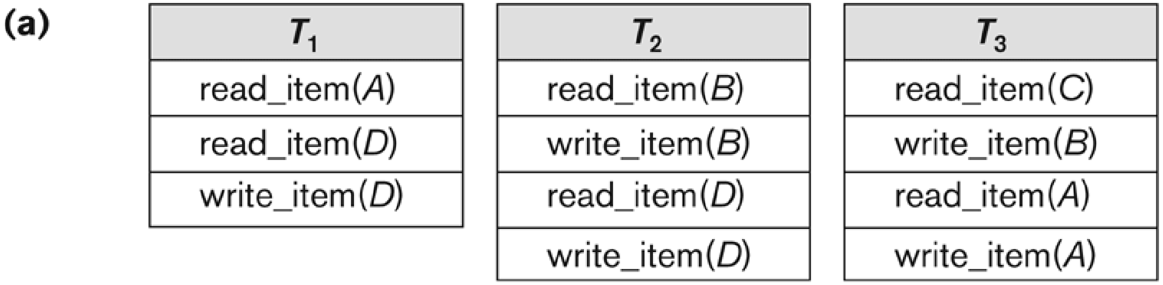
- Data items to be modified are first stored into database cache by the Cache Manager (CM) and after modification they are flushed (written) to the disk.
- The flushing is controlled by **Modified** and **Pin-Unpin** bits.
  - **Pin-Unpin**: Instructs the operating system not to flush the data item.
  - **Modified**: Indicates the AFIM of the data item.

Database recovery techniques

Transaction **Roll-back (Undo)** and **Roll-Forward (Redo)**

- To maintain atomicity, a transaction's operations are redone or undone.
  - **Undo**: Restore all BFIMs on to disk (Remove all AFIMs).
  - **Redo**: Restore all AFIMs on to disk.
- Database recovery is achieved either by performing only Undos or only Redos or by a combination of the two.
- These operations are recorded in the log as they happen.

Database recovery techniques



(a)

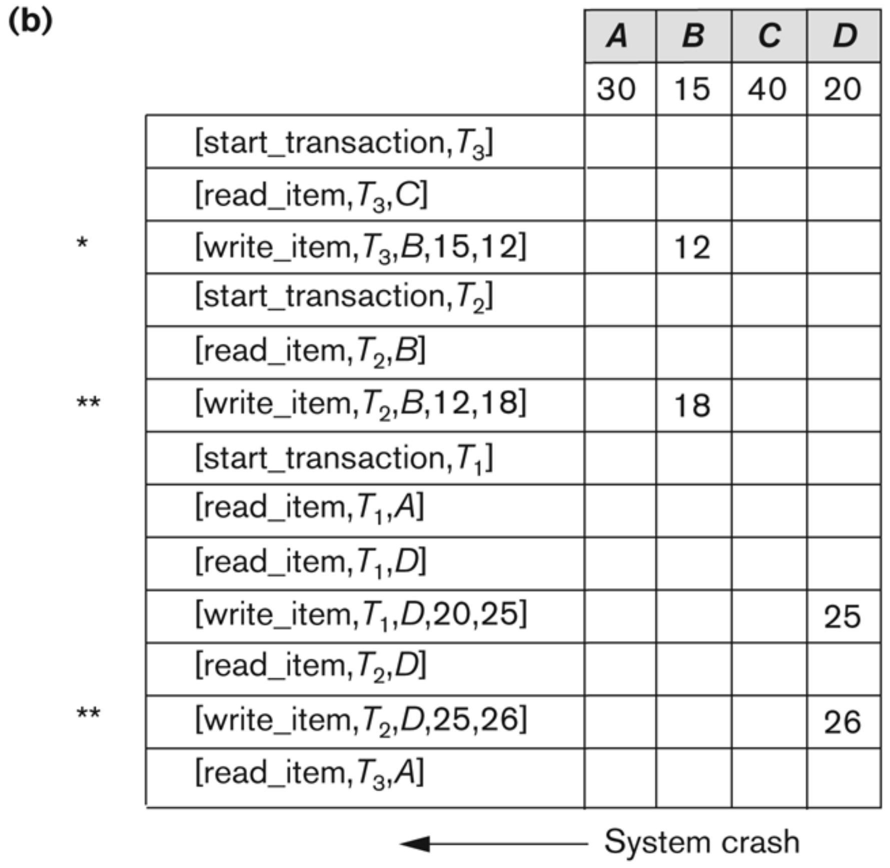
T_1
read_item(A)
read_item(D)
write_item(D)

T_2
read_item(B)
write_item(B)
read_item(D)
write_item(D)

T_3
read_item(C)
write_item(B)
read_item(A)
write_item(A)

Figure 19.1
Illustrating cascading rollback (a process that never occurs in strict or cascadeless schedules). (a) The read and write operations of three transactions. (b) System log at point of crash. (c) Operations before the crash.

Database recovery techniques



(b)

		A	*B*	*C*	*D*
		30	15	40	20
	[start_transaction,T_3]				
	[read_item,T_3,*C*]				
*	[write_item,T_3,*B*,15,12]		12		
	[start_transaction,T_2]				
	[read_item,T_2,*B*]				
**	[write_item,T_2,*B*,12,18]		18		
	[start_transaction,T_1]				
	[read_item,T_1,*A*]				
	[read_item,T_1,*D*]				
	[write_item,T_1,*D*,20,25]				25
	[read_item,T_2,*D*]				
**	[write_item,T_2,*D*,25,26]				26
	[read_item,T_3,*A*]				

System crash

Figure 19.1
Illustrating cascading rollback (a process that never occurs in strict or cascadeless schedules). (a) The read and write operations of three transactions. (b) System log at point of crash. (c) Operations before the crash.

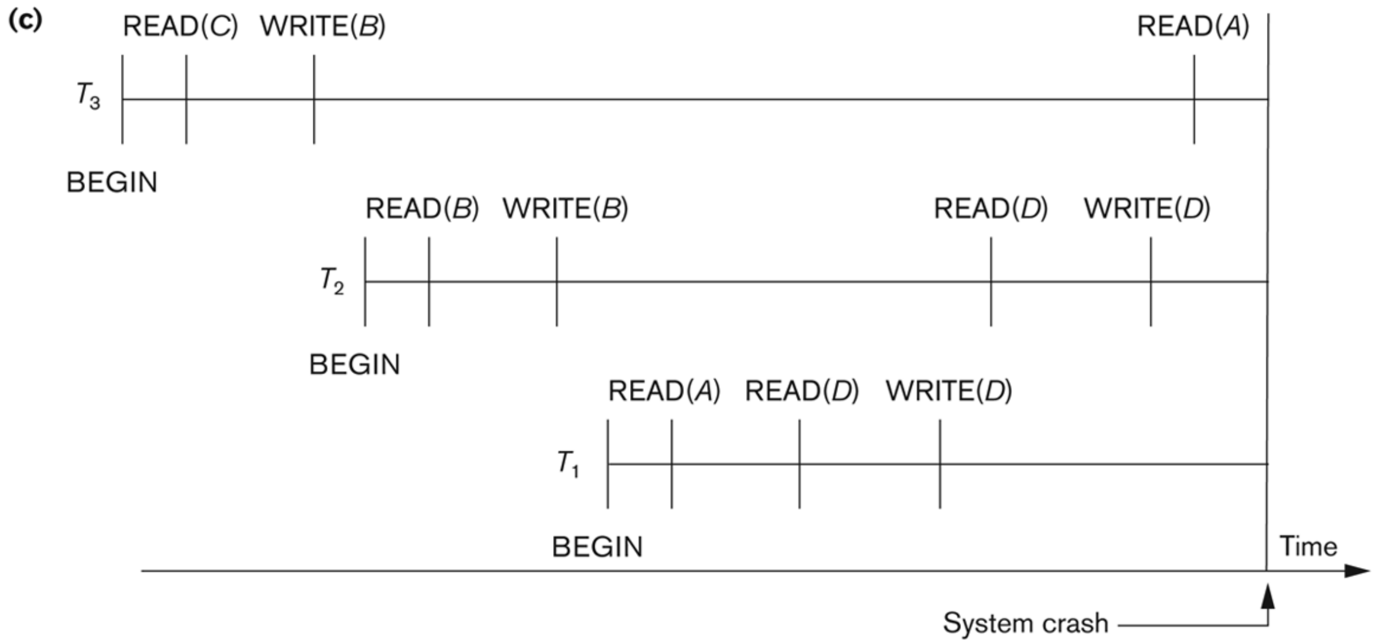
* T_3 is rolled back because it did not reach its commit point.

** T_2 is rolled back because it reads the value of item *B* written by T_3.

Database recovery techniques

Roll-back: One execution of T1, T2 and T3 as recorded in the log.

Figure 19.1
Illustrating cascading rollback (a process that never occurs in strict or cascadeless schedules). (a) The read and write operations of three transactions. (b) System log at point of crash. (c) Operations before the crash.


(c)
READ(C)
WRITE(B)
READ(A)
T_3
BEGIN
READ(B)
WRITE(B)
READ(D)
WRITE(D)
T_2
BEGIN
READ(A)
READ(D)
WRITE(D)
T_1
BEGIN
Time
System crash

Database recovery techniques

Write-Ahead Logging

- When **in-place** update (immediate or deferred) is used then log is necessary for recovery and it must be available to recovery manager. This is achieved by **Write-Ahead Logging (WAL)** protocol. WAL states that
  - **For Undo**: Before a data item's AFIM is flushed to the database disk (overwriting the BFIM) its BFIM must be written to the log and the log must be saved on a stable store (log disk).
  - **For Redo**: Before a transaction executes its commit operation, all its AFIMs must be written to the log and the log must be saved on a stable store.

Database recovery techniques

Checkpointing

- Time to time (randomly or under some criteria) the database flushes its buffer to database disk to minimize the task of recovery.
- The following steps defines a checkpoint operation:
  1. Suspend execution of transactions temporarily.
  2. Force write modified buffer data to disk.
  3. Write a [checkpoint] record to the log, save the log to disk.
  4. Resume normal transaction execution.
- During recovery redo or undo is required to transactions appearing after [checkpoint] record.

Database recovery techniques

Steal/No-Steal and Force/No-Force

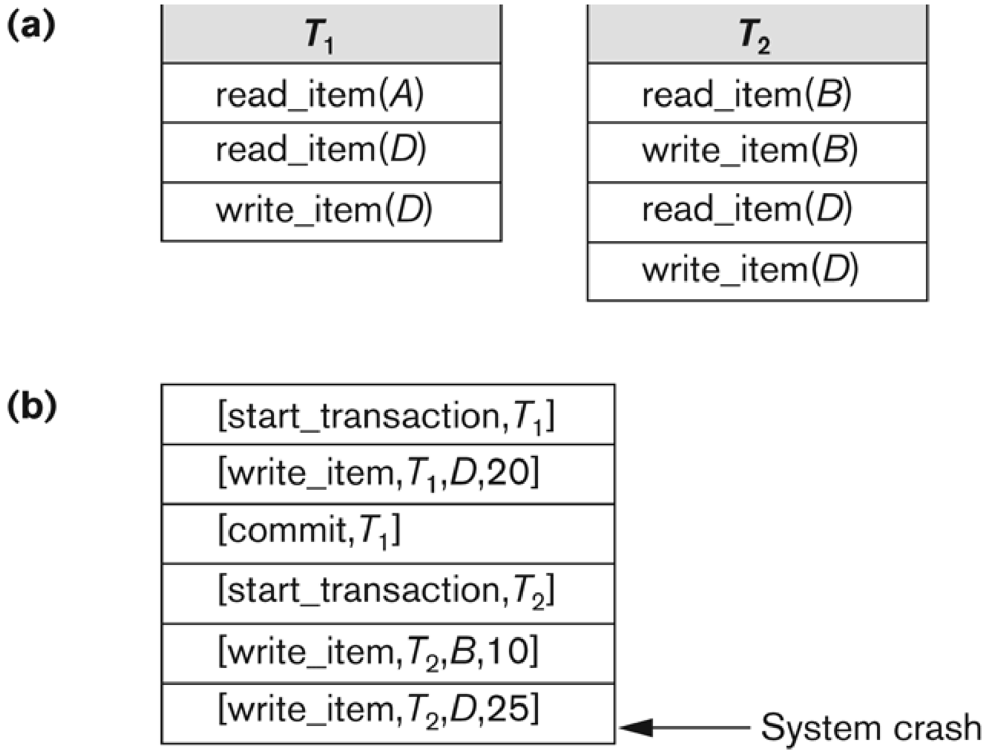
- Possible ways for flushing database cache to database disk:
  1. Steal: Cache can be flushed before transaction commits.
  2. No-Steal: Cache cannot be flushed before transaction commit.
  3. Force: Cache is immediately flushed (forced) to disk.
  4. No-Force: Cache is deferred until transaction commits
- These give rise to four different ways for handling recovery:
  - Steal/No-Force (Undo/Redo)
  - Steal/Force (Undo/No-redo)
  - No-Steal/No-Force (Redo/No-undo)
  - No-Steal/Force (No-undo/No-redo)

Recovery Scheme

- Deferred Update (No Undo/Redo)
  - The data update goes as follows:
  - A set of transactions records their updates in the log.
  - At commit point under WAL scheme, these updates are saved on database disk.
  - After reboot from a failure the log is used to redo all the transactions affected by this failure.
  - No undo is required because no AFIM is flushed to the disk before a transaction commits.

Database recovery techniques

- Deferred Update in a single-user system
  There is no concurrent data sharing in a single user system.
- The data update goes as follows:
  - A set of transactions records their updates in the log.
  - At commit point under WAL scheme these updates are saved on database disk.
- After reboot from a failure the log is used to redo all the transactions affected by this failure.
- No undo is required because no AFIM is flushed to the disk before a transaction commits.



(a)

T_1
read_item(A)
read_item(D)
write_item(D)

T_2
read_item(B)
write_item(B)
read_item(D)
write_item(D)

(b)

[start_transaction, T_1]
[write_item, T_1, D, 20]
[commit, T_1]
[start_transaction, T_2]
[write_item, T_2, B, 10]
[write_item, T_2, D, 25]

System crash

The [write_item,...] operations of T_1 are redone.

T_2 log entries are ignored by the recovery process.

Figure 19.2
An example of recovery using deferred update in a single-user environment. (a) The READ and WRITE operations of two transactions. (b) The system log at the point of crash.

innovate achieve lead

Deferred Update with concurrent users

- This environment requires some concurrency control mechanism to guarantee **isolation** property of transactions. In a system recovery, transactions which were recorded in the log after the last checkpoint were **redone**. The recovery manager may scan some of the transactions recorded before the checkpoint to get the AFIMs.

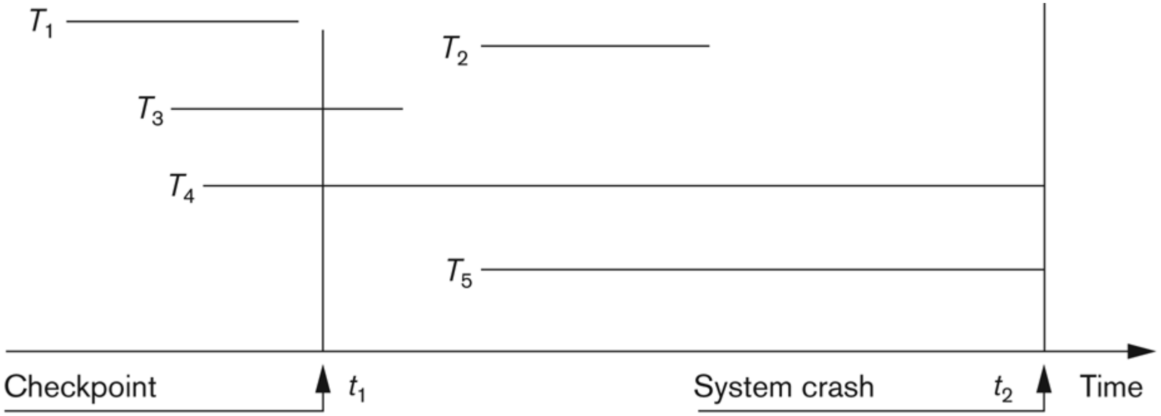
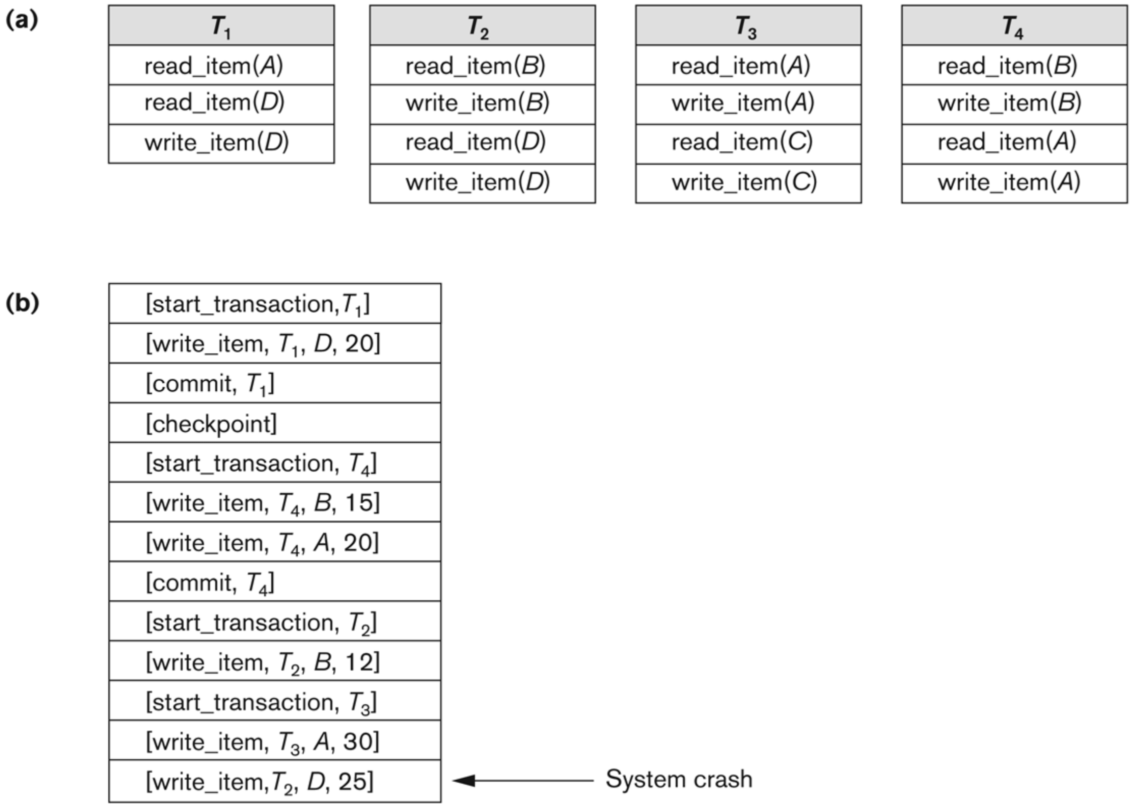


Figure 19.3
An example of recovery in a multi-user environment



(a)

T_1	T_2	T_3	T_4
read_item(A)	read_item(B)	read_item(A)	read_item(B)
read_item(D)	write_item(B)	write_item(A)	write_item(B)
write_item(D)	read_item(D)	read_item(C)	read_item(A)
	write_item(D)	write_item(C)	write_item(A)

(b)

[start_transaction, T_1]	
[write_item, T_1, D, 20]	
[commit, T_1]	
[checkpoint]	
[start_transaction, T_4]	
[write_item, T_4, B, 15]	
[write_item, T_4, A, 20]	
[commit, T_4]	
[start_transaction, T_2]	
[write_item, T_2, B, 12]	
[start_transaction, T_3]	
[write_item, T_3, A, 30]	
[write_item, T_2, D, 25]	← System crash

T_2 and T_3 are ignored because they did not reach their commit points.

T_4 is redone because its commit point is after the last system checkpoint.

Figure 19.4
An example of recovery using deferred update with concurrent transactions.
(a) The READ and WRITE operations of four transactions. (b) System log at the point of crash.

Deferred Update with concurrent users

- Two tables are required for implementing this protocol:
  - **Active table**: All active transactions are entered in this table.
  - **Commit table**: Transactions to be committed are entered in this table.
- During recovery, all transactions of the **commit** table are redone and all transactions of **active** tables are ignored since none of their AFIMs reached the database.
- It is possible that a **commit** table transaction may be **redone** twice but this does not create any inconsistency because of a redone is "**idempotent**", that is, one redone for an AFIM is equivalent to multiple redone for the same AFIM.

Recovery Techniques Based on Immediate Update

- **Undo/No-redo Algorithm**
  - In this algorithm AFIMs of a transaction are flushed to the database disk under WAL before it commits.
  - For this reason the recovery manager **undoes** all transactions during recovery.
  - No transaction is **redone**.
  - It is possible that a transaction might have completed execution and ready to commit but this transaction is also **undone**.

Recovery Techniques Based on Immediate Update

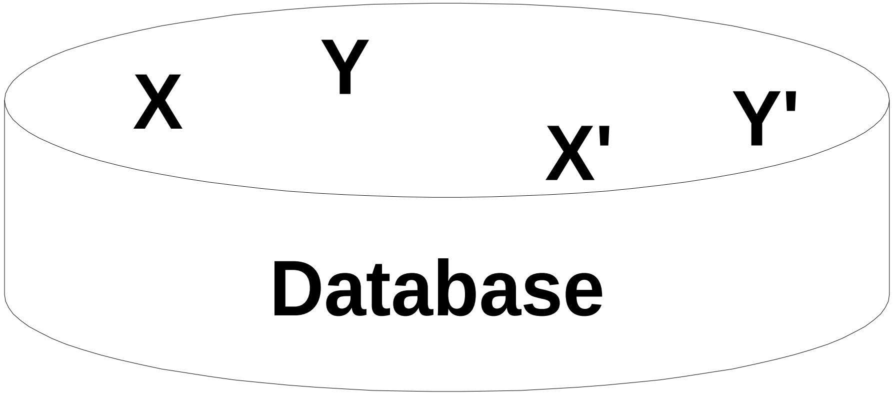
- **Undo/Redo Algorithm (Single-user** environment)
  - Recovery schemes of this category apply **undo** and also **redo** for recovery.
  - In a single-user environment no concurrency control is required but a log is maintained under WAL.
  - Note that at any time there will be one transaction in the system and it will be either in the commit table or in the active table.
  - The recovery manager performs:
    - **Undo** of a transaction if it is in the **active** table.
    - **Redo** of a transaction if it is in the **commit** table.

Recovery Techniques Based on Immediate Update

- **Undo/Redo Algorithm** (**Concurrent** execution)
- Recovery schemes of this category applies **undo** and also **redo** to recover the database from failure.
- In concurrent execution environment a concurrency control is required and log is maintained under WAL.
- Commit table records transactions to be committed and active table records active transactions.
- To minimize the work of the recovery manager checkpointing is used.
- The recovery performs:
  - **Undo** of a transaction if it is in the **active** table.
  - **Redo** of a transaction if it is in the **commit** table.

Shadow Paging

- The AFIM does not overwrite its BFIM but recorded at another place on the disk. Thus, at any time a data item has AFIM and BFIM (Shadow copy of the data item) at two different places on the disk.



X and Y: Shadow copies of data items
X' and Y': Current copies of data items

Shadow Paging

- To manage access of data items by concurrent transactions two directories (current and shadow) are used.
  - The directory arrangement is illustrated below. Here a page is a data item.

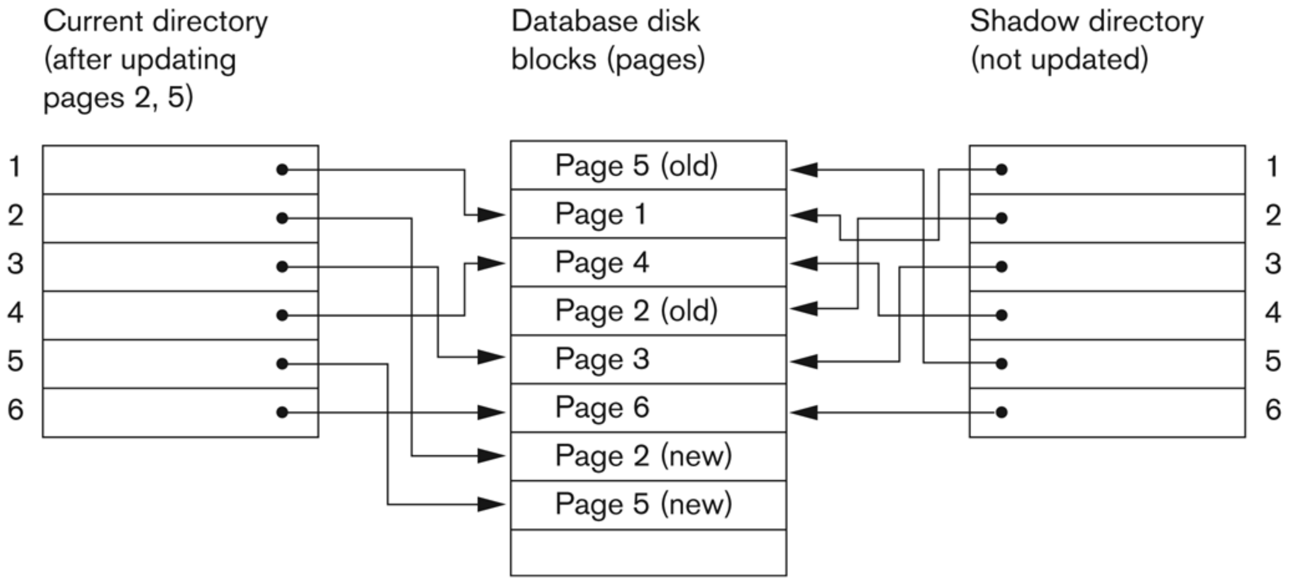

Current directory (after updating pages 2, 5)
Database disk blocks (pages)
Shadow directory (not updated)
1
2
3
4
5
6
Page 5 (old)
Page 1
Page 4
Page 2 (old)
Page 3
Page 6
Page 2 (new)
Page 5 (new)
1
2
3
4
5
6

Figure 19.5
An example of shadow paging.

Thank you….

Database Design

BITS Pilani

Pilani Campus

innovate achieve lead

BITS Pilani
Pilani Campus

Triggers in SQLite

SQLite Trigger

SQLite **Triggers** are database callback functions, which are automatically performed/invoked when a specified database event occurs. Following are the important points about SQLite triggers -

- SQLite trigger may be specified to fire whenever a DELETE, INSERT or UPDATE of a particular database table occurs or whenever an UPDATE occurs on one or more specified columns of a table.
- At this time, SQLite supports only FOR EACH ROW triggers, not FOR EACH STATEMENT triggers.
- Hence, explicitly specifying FOR EACH ROW is optional.
- Both the WHEN clause and the trigger actions may access elements of the row being inserted, deleted, or updated using references of the form **NEW.column-name** and **OLD.column-name**, where column-name is the name of a column from the table that the trigger is associated with.

SQLite Trigger

- If a WHEN clause is supplied, the SQL statements specified are only executed for rows for which the WHEN clause is true.
- If no WHEN clause is supplied, the SQL statements are executed for all rows.
- The BEFORE or AFTER keyword determines when the trigger actions will be executed relative to the insertion, modification, or removal of the associated row.
- Triggers are automatically dropped when the table that they are associated with is dropped.
- The table to be modified must exist in the same database as the table or view to which the trigger is attached and one must use just **tablename** not **database.tablename**.
- A special SQL function RAISE() may be used within a trigger-program to raise an exception.

When do we need SQLite triggers

An SQLite trigger is a named database object that is executed automatically when an INSERT, UPDATE or DELETE statement is issued against the associated table.

You often use triggers to enable sophisticated auditing.

For example, you want to log the changes in the sensitive data such as salary and address whenever it changes.

In addition, you use triggers to enforce complex business rules centrally at the database level and prevent invalid transactions.

SQLite Trigger - Syntax

Following is the basic syntax of creating a **trigger**.

CREATE TRIGGER trigger_name [BEFORE|AFTER] event_name ON table_name

BEGIN -- Trigger logic goes here....END;

Here, **event_name** could be *INSERT, DELETE,* and *UPDATE* database operation on the mentioned table **table_name**.

You can optionally specify FOR EACH ROW after table name.

Following is the syntax for creating a trigger on an UPDATE operation on one or more specified columns of a table.

CREATE TRIGGER trigger_name [BEFORE|AFTER] UPDATE OF column_name ON table_nameBEGIN   -- Trigger logic goes here....END;

SQLite CREATE TRIGGER statement

To create a new trigger in SQLite, you use the CREATE TRIGGER statement as follows:

```
CREATE TRIGGER [IF NOT EXISTS] trigger_name
   [BEFORE|AFTER|INSTEAD OF] [INSERT|UPDATE|DELETE]
   ON table_name
   [WHEN condition]
BEGIN
 statements;
END;
```

SQLite CREATE TRIGGER statement

In this syntax:

- First, specify the name of the trigger after the CREATE TRIGGER keywords.
- Next, determine when the trigger is fired such as BEFORE, AFTER, or INSTEAD OF. You can create BEFORE and AFTER triggers on a table. However, you can only create an INSTEAD OF trigger on a view.
- Then, specify the event that causes the trigger to be invoked such as INSERT, UPDATE, or DELETE.
- After that, indicate the table to which the trigger belongs.
- Finally, place the trigger logic in the BEGIN END block, which can be any valid SQL statements.

SQLite CREATE TRIGGER statement

If you combine the time when the trigger is fired and the event that causes the trigger to be fired, you have a total of 9 possibilities:

- BEFORE INSERT
- AFTER INSERT
- BEFORE UPDATE
- AFTER UPDATE
- BEFORE DELETE
- AFTER DELEE
- INSTEAD OF INSERT
- INSTEAD OF DELETE
- INSTEAD OF UPDATE

SQLite - Trigger

Suppose you use a UPDATE statement to update 10 rows in a table, the trigger that associated with the table is fired 10 times. This trigger is called FOR EACH ROW trigger. If the trigger associated with the table is fired one time, we call this trigger a FOR EACH STATEMENT trigger.

As of version 3.9.2, SQLite only supports FOR EACH ROW triggers. It has not yet supported the FOR EACH STATEMENT triggers.

If you use a condition in the WHEN clause, the trigger is only invoked when the condition is true. In case you omit the WHEN clause, the trigger is executed for all rows.

Notice that if you drop a table, all associated triggers are also deleted. However, if the trigger references other tables, the trigger is not removed or changed if other tables are removed or updated.

For example, a trigger references to a table named people, you drop the people table or rename it, you need to manually change the definition of the trigger.

SQLite - Trigger

You can access the data of the row being inserted, deleted, or updated using the OLD and NEW references in the form: OLD.column_name and NEW.column_name.

the OLD and NEW references are available depending on the event that causes the trigger to be fired.

The following table illustrates the rules.:

Action	Reference
INSERT	NEW is available
UPDATE	Both NEW and OLD are available
DELETE	OLD is available

SQLite Trigger - Examples

Let us consider a case where we want to keep audit trail for every record being inserted in COMPANY table, which we create newly as follows (Drop COMPANY table if you already have it).

```
sqlite> CREATE TABLE COMPANY(   ID INT PRIMARY KEY     NOT NULL,   NAME
    TEXT    NOT NULL,   AGE            INT     NOT NULL,   ADDRESS        CHAR(50),   SALARY
        REAL);
```

To keep audit trial, we will create a new table called AUDIT where the log messages will be inserted, whenever there is an entry in COMPANY table for a new record.

```
sqlite> CREATE TABLE AUDIT(   EMP_ID INT NOT NULL,   ENTRY_DATE TEXT NOT
    NULL);
```

Here, ID is the AUDIT record ID, and EMP_ID is the ID which will come from COMPANY table and DATE will keep timestamp when the record will be created in COMPANY table.

SQLite Trigger - Examples

Now let's create a trigger on COMPANY table as follows -

```
sqlite> CREATE TRIGGER audit_log AFTER INSERT ON COMPANYBEGIN   INSERT INTO AUDIT(EMP_ID, ENTRY_DATE) VALUES (new.ID, datetime('now'));END;
```

Now, we will start actual work, Let's start inserting record in COMPANY table which should result in creating an audit log record in AUDIT table. Create one record in COMPANY table as follows -

```
sqlite> INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY)VALUES (1, 'Paul', 32, 'California', 20000.00 );
```

This will create one record in COMPANY table, which is as follows -

```
ID          NAME        AGE         ADDRESS     SALARY
----------  ----------  ----------  ----------  ----------
1           Paul        32          California  20000.0
```

Same time, one record will be created in AUDIT table. This record is the result of a trigger, which we have created on INSERT operation in COMPANY table.

Similarly, you can create your triggers on UPDATE and DELETE operations based on your requirements.

```
EMP_ID      ENTRY_DATE
----------  -------------------
1           2013-04-05 06:26:00
```

SQLite Trigger - Listing Triggers

You can list down all the triggers from **sqlite_master** table as follows -

```
sqlite> SELECT name FROM sqlite_masterWHERE type = 'trigger';
```

The above SQLite statement will list down only one entry as follows -

```
name----------audit_log
```

If you want to list down triggers on a particular table, then use AND clause with table name as follows -

```
sqlite> SELECT name FROM sqlite_master WHERE type = 'trigger' AND tbl_name = 'COMPANY';
```

The above SQLite statement will also list down only one entry as follows -

```
name----------audit_log
```

Dropping Triggers

Following is the DROP command, which can be used to drop an existing trigger.

```
sqlite> DROP TRIGGER trigger_name;
```

SQLite Trigger - Examples

Let’s create a new table called leads to store all business leads of the company.

```
CREATE TABLE leads (
    id integer PRIMARY KEY,
    first_name text NOT NULL,
    last_name text NOT NULL,
    phone text NOT NULL,
    email text NOT NULL,
    source text NOT NULL
);
```

SQLite BEFORE INSERT trigger example

1) SQLite BEFORE INSERT trigger example

Suppose you want to validate the email address before inserting a new lead into the leads table. In this case, you can use a BEFORE INSERT trigger.

First, create a BEFORE INSERT trigger as follows:

```
CREATE TRIGGER validate_email_before_insert_leads
   BEFORE INSERT ON leads
BEGIN
   SELECT
      CASE
      WHEN NEW.email NOT LIKE '%_@__%.__%' THEN
        RAISE (ABORT,'Invalid email address')
       END; END;
```

SQLite BEFORE INSERT trigger example

We used the NEW reference to access the email column of the row that is being inserted.

To validate the email, we used the LIKE operator to determine whether the email is valid or not based on the email pattern. If the email is not valid, the RAISE function aborts the insert and issues an error message.

Second, insert a row with an invalid email into the leads table.

INSERT INTO leads (id,first_name,last_name,email,phone,source)

VALUES(1,'John','Doe','jjj','4089009334','New Your');SQLite issued an error: “Invalid email address” and aborted the execution of the insert.

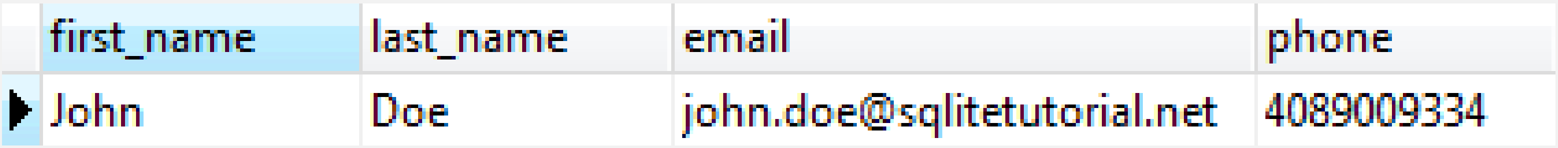
SQLite BEFORE INSERT trigger example

Third, insert a row with a valid email.

```
INSERT INTO leads (first_name, last_name, email, phone)

VALUES ('John', 'Doe', 'john.doe@sqlitetutorial.net', '4089009334');
```

Because the email is valid, the insert statement executed successfully.

```
SELECT  first_name,last_name,            email,    phone FROM leads;
```



first_name	last_name	email	phone
John	Doe	john.doe@sqlitetutorial.net	4089009334

SQLite AFTER UPDATE trigger example

SQLite AFTER UPDATE trigger example

The phones and emails of the leads are so important that you can’t afford to lose this information. For example, someone accidentally updates the email or phone to the wrong ones or even delete it.

To protect this valuable data, you use a trigger to log all changes which are made to the phone and email.

First, create a new table called lead_logs to store the historical data.

CREATE TABLE lead_logs (id INTEGER PRIMARY KEY, old_id int, new_id int, old_phone text, new_phone text,

old_email text, new_email text, user_action text, created_at text);

SQLite AFTER UPDATE trigger example

Second, create an AFTER UPDATE trigger to log data to the lead_logs table whenever there is an update in the email or phone column.

```
CREATE TRIGGER log_contact_after_update    AFTER UPDATE ON leads    WHEN old.phone <>
    new.phone

      OR old.email <> new.email

BEGIN

    INSERT INTO lead_logs (old_id,new_id,old_phone,    new_phone,old_email,new_email,user_action,
               created_at          )VALUES (old.id,new.id,old.phone,new.phone,          old.email,
    new.email,          'UPDATE',

               DATETIME('NOW')

    ) ;

END;
```

SQLite AFTER UPDATE trigger example

You notice that in the condition in the WHEN clause specifies that the trigger is invoked only when there is a change in either email or phone column.

Third, update the last name of John from Doe to Smith.

```
UPDATE leads SET    last_name = 'Smith’ WHERE   id = 1;
```

The trigger log_contact_after_update was not invoked because there was no change in email or phone.

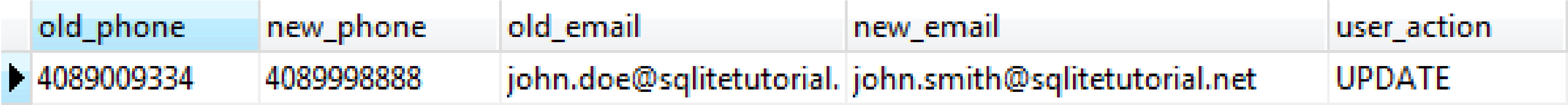
Fourth, update both email and phone of John to the new ones.

```
UPDATE leads SET    phone = '4089998888',   email = 'john.smith@sqlitetutorial.net’ WHERE   id = 1;
```

SQLite AFTER UPDATE trigger example

If you check the log table, you will see there is a new entry there.

SELECT old_phone, new_phone, old_email, new_email, user_action FROM lead_logs;



	old_phone	new_phone	old_email	new_email	user_action
▶	4089009334	4089998888	john.doe@sqlitetutorial.	john.smith@sqlitetutorial.net	UPDATE

Exercise

Develop the AFTER INSERT and AFTER DELETE triggers to log the data in the lead_logs table as an excercise.

SQLite Trigger – Example 3

Assuming that customer records are stored in the "customers" table, and that order records are stored in the "orders" table, the following UPDATE trigger ensures that all associated orders are redirected when a customer changes his or her address:

```
CREATE TRIGGER update_customer_address UPDATE OF address ON customers
  BEGIN
    UPDATE orders SET address = new.address WHERE customer_name = old.name;
  END;
```

With this trigger installed, executing the statement:

```
UPDATE customers SET address = '1 Main St.' WHERE name = 'Jack Jones';
```

causes the following to be automatically executed:

```
UPDATE orders SET address = '1 Main St.' WHERE customer_name = 'Jack Jones';
```

SQLite Trigger – Example 3

For an example of an INSTEAD OF trigger, consider the following schema:

```
CREATE TABLE customer(
  cust_id INTEGER PRIMARY KEY,
  cust_name TEXT,
  cust_addr TEXT
);
CREATE VIEW customer_address AS SELECT cust_id, cust_addr FROM customer;
CREATE TRIGGER cust_addr_chng
INSTEAD OF UPDATE OF cust_addr ON customer_address
BEGIN
  UPDATE customer SET cust_addr=NEW.cust_addr
   WHERE cust_id=NEW.cust_id;  END;
```

SQLite Trigger – Example 3

With the schema above, a statement of the form:

```
UPDATE customer_address SET cust_addr=$new_address WHERE cust_id=$cust_id;


UPDATE customer_address SET cust_addr=’Mumbai’ WHERE cust_id=1;
```

Causes the customer.cust_addr field to be updated for a specific customer entry that has customer.cust_id equal to the $cust_id parameter.

Note how the values assigned to the view are made available as field in the special "NEW" table within the trigger body.

SQLite3 - Java

BITS Pilani
Pilani Campus

BITS Pilani
Pilani Campus

Java Applications – Java Database connectivity

Installation - SQLite3

- **Step 1** - Go to SQLite download page, and download precompiled binaries from Windows section.
- **Step 2** - Download sqlite-shell-win32-*.zip and sqlite-dll-win32-*.zip zipped files.
- **Step 3** - Create a folder C:\>sqlite and unzip above two zipped files in this folder, which will give you sqlite3.def, sqlite3.dll and sqlite3.exe files.
- **Step 4** - Add C:\>sqlite in your PATH environment variable and finally go to the command prompt and issue sqlite3 command, which should display the following result.
- C:\>sqlite3
- SQLite version 3.7.15.2 2013-01-09 11:53:05
- Enter ".help" for instructions
- Enter SQL statements terminated with a ";"
- sqlite>

Installation – sqlite-jdbc driver

- Download latest version of *sqlite-jdbc-(VERSION).jar* from sqlite-jdbc repository.
- Add downloaded jar file *sqlite-jdbc-(VERSION).jar* in your class path, or you can use it along with -classpath option as explained in the following examples.

Connect to database

```
import java.sql.*;
public class SQLiteJDBC {
  public static void main( String args[] ) {
      Connection c = null;
      try {
      Class.forName("org.sqlite.JDBC");
         c = DriverManager.getConnection("jdbc:sqlite:Lab2.db");
      } catch ( Exception e ) {
         System.err.println( e.getClass().getName() + ": " + e.getMessage() );
         System.exit(0);
      }
      System.out.println("Opened database successfully");
   } }
```

Compile and execute as follows:

```
C:>javac SQLiteJDBC.java
C:>java -classpath ".;sqlite-jdbc-3.27.2.1.jar" SQLiteJDBC
```

Create Table

```
import java.sql.*;
public class CreateTableCompany {
  public static void main( String args[] ) {
    Connection c = null;
    Statement stmt = null;
    try {
      Class.forName("org.sqlite.JDBC");
      c = DriverManager.getConnection("jdbc:sqlite:Lab2.db");
      System.out.println("Opened database successfully");
      stmt = c.createStatement();
      String sql = "CREATE TABLE COMPANYNew " +
                "(ID INT PRIMARY KEY     NOT NULL," +
                " NAME           TEXT    NOT NULL, " +
                " AGE            INT     NOT NULL, " +
                " ADDRESS        CHAR(50), " +
                " SALARY         REAL)";
stmt.executeUpdate(sql);
      stmt.close();
      c.close();
    } catch ( Exception e ) {
      System.err.println( e.getClass().getName()
+ ": " + e.getMessage() );
      System.exit(0);
    }
    System.out.println("Table created
successfully");
  }
}
```

Insert data into the created table

```
import java.sql.*;
public class InsertIntoCompany {
   public static void main( String args[] ) {
      Connection c = null;
      Statement stmt = null;
      try {
         Class.forName("org.sqlite.JDBC");
         c = DriverManager.getConnection("jdbc:sqlite:Lab2.db");
         c.setAutoCommit(false);
         System.out.println("Opened database successfully");
         stmt = c.createStatement();
         String sql = "INSERT INTO COMPANYNew (ID,NAME,AGE,ADDRESS,SALARY) " +
                        "VALUES (1, 'Paul', 32, 'California', 20000.00 );";
         stmt.executeUpdate(sql);
```

Insert data into the created table

```
sql = "INSERT INTO COMPANYNew (ID,NAME,AGE,ADDRESS,SALARY) " +"VALUES (2, 'Allen', 25, 'Texas',
    15000.00 );";
      stmt.executeUpdate(sql);
      sql = "INSERT INTO COMPANYNew (ID,NAME,AGE,ADDRESS,SALARY) " + "VALUES (3, 'Teddy', 23,
    'Norway', 20000.00 );";
      stmt.executeUpdate(sql);
      sql = "INSERT INTO COMPANYNew (ID,NAME,AGE,ADDRESS,SALARY) " + "VALUES (4, 'Mark', 25, 'Rich-
    Mond ', 65000.00 );";
      stmt.executeUpdate(sql);
      stmt.close();
      c.commit();
      c.close();
    } catch ( Exception e ) {
      System.err.println( e.getClass().getName() + ": " + e.getMessage() );
      System.exit(0);
    }
    System.out.println("Records created successfully");   } }
```

Fetch data from created table

```
import java.sql.*;
public class SelectCompany {
  public static void main( String args[] ) {
   Connection c = null;
   Statement stmt = null;
   try {
      Class.forName("org.sqlite.JDBC");
      c = DriverManager.getConnection("jdbc:sqlite:test.db");
      c.setAutoCommit(false);
      System.out.println("Opened database successfully");
      stmt = c.createStatement();
      ResultSet rs = stmt.executeQuery( "SELECT * FROM COMPANYNew;" );
```

Fetch data from created table

```
while ( rs.next() ) {
        int id = rs.getInt("id");
        String  name = rs.getString("name");
        int age  = rs.getInt("age");
        String  address = rs.getString("address");
        float salary = rs.getFloat("salary");
        System.out.println( "ID = " + id );
        System.out.println( "NAME = " + name );
        System.out.println( "AGE = " + age );
        System.out.println( "ADDRESS = " + address );
        System.out.println( "SALARY = " + salary );
        System.out.println();
     }
```

Fetch data from created table

```
rs.close();
      stmt.close();
      c.close();
   } catch ( Exception e ) {
      System.err.println( e.getClass().getName() + ": " + e.getMessage() );
      System.exit(0);
   }
   System.out.println("Operation done successfully");
  }
}
```

Thank you

Database Systems and Applications

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

Contact Session 14:

Database Security, Access Control Mechanisms, to SQL Query Processing

Objectives:

- To understand Database Security with Access control schemes;
- To understand basics of SQL query optimization
- To understand optimization algorithms to perform certain operations.

Contents:

- Introduction to Database Security
- Access Control Mechanisms
- Introduction to SQL Query Processing
- Algorithms for Sorting and Joining
- Algorithms for SELECT, PROJECT and Set operations
- Algorithms for Aggregation and Grouping
- query optimization,

Let's have a quick recap!!

- Concurrency control schemes
- Implementing serializability with locks and timestamps;
- Deadlocks with suitable examples

Main Types of Database Security:

- Enforce security of portions of a database against unauthorized access
  - Database Security and Authorization Subsystem
- Prevent unauthorized persons from accessing the system itself
  - Access Control
- Control the access to databases
  - Database Security
- Protect sensitive data that is being transmitted via some type of communications
  - Data Encryption

Database Security and Authorization Subsystem

- **Discretionary Security Mechanisms**

  - concerned with defining, modeling, and enforcing access to information

- **Mandatory Security Mechanisms** for Multilevel Security

  - requires that data items and users are assigned to certain security labels.

Mandatory access control (MAC) is a model of access control where the operating system provides users with access based on data confidentiality and user clearance levels. In this model, access is granted on a need to know basis: users have to prove a need for information before gaining access.

MAC is considered the most secure of all access control models. Access rules in this model are manually defined by system administrators and strictly enforced by the operating system or security kernel. Regular users can’t alter security attributes even for data they’ve created.

Origin: https://www.ekransystem.com/en/blog/mac-vs-dac, © Ekran System

Database Security and Authorization Subsystem

Discretionary access control is an identity-based access control model that provides users with a certain amount of control over their data.

Data owners (document creators or any users authorized to control data) can define access permissions for specific users or groups of users.

In other words, whom to give access to and what privileges to grant are decided at the resource owner’s discretion.

Access permissions for each piece of data are stored in an access control list (ACL).

An administrator creates this list when a user grants access to somebody.

The list can be generated automatically.

An ACL includes users and groups that may access data and the levels of access they have.

A system administrator can also enforce an ACL. In this case, the ACL acts as a security policy, and regular users can’t edit or overrule it.

Mandatory Access Control

Elements:

OBJECTS

(file, database, port, etc.)

CLASSIFICATIONS

--class(object)--

SUBJECTS

(Users)

CLEARANCE

--clear(subject)--

Levels: Top Secret, Secret, Confidential, Unclassified

Mandatory Access Control

Rules:

- Simple Property: subject *s* is allowed to read data item *d* if clear(s) ≥ class(d)
- *-property: subject *s* is allowed to write data item *d* if clear(s) ≤ class(d)

(Less intuitive / reasonable rule)

- Simple Property protects information from unauthorized access
- *-property protects data from contamination or unauthorized modification

Preventing SQL Injection

- SQL injection is **a code injection technique that might destroy your database**. SQL injection is one of the most common web hacking techniques.

SQL Injection attacks (or SQLi) alter SQL queries, injecting malicious code by exploiting application vulnerabilities.

Successful SQLi attacks allow attackers to modify database information, access sensitive data, execute admin tasks on the database, and recover files from the system.

In some cases attackers can issue commands to the underlying database operating system.

- The severe impact of these attacks makes it critical for developers to adopt practices that prevent SQL injection, such as parameterized queries, stored procedures, and rigorous input validation.

Preventing SQL Injection

- To prevent SQL injection attacks, use prepared statements (instead of creating query strings from input parameters)
  - PreparedStatement pstmt= conn.prepareStatement(
    "select balance from account where account_number =?“);
    pstmt.setString(1,accnt_number);
    pstmt.execute();
    - (assume that conn is an already open connection to the database)
- Alternatives:
  - use stored procedures
  - use a function that removes special characters (such as quotes) from strings

- Refer for SQL injections: https://portswigger.net/web-security/sql-injection#:~:text=Some%20common%20SQL%20injection%20examples,data%20from%20different%20database%20tables
  .

Protecting from Programmers/DBA

- Have password to database, can update anything!
  - Digital signatures by end users can help in some situations.
  - A digital signature confirms that any macros, code modules, and other executable components in the database originated with the signer and that no one has altered them since the database was signed. This helps people who use the database decide whether to trust it and its content.
    - E.g. low update rate data such as land records, birth/death data
- Application program has database password
  - Seize control of the application program ➔ can do anything to the database
  - Solution:
    - Don’t give database password to development team
    - keep password in a configuration file on live server, accessible to only a few system administrators
- Ongoing research on trusted applications
- Applications with special privileges that perform security-related functions are called trusted applications.
  - E.g. OS computes checksum on application to verify corruption
  - Allows file-system access only to trusted applications

Detecting Corruption

- Audit trails: record of all (update) activity on the database: who did what, when
  - Application-level audit trail
    - Helps detect fraudulent activities by users
    - Independent audit section to check all updates
    - BUT: DBAs can bypass this level
      - E.g. audit trail apparently deleted in New Delhi auto-rickshaw license case by malicious users with DBA access
  - Database level audit trail
    - Database needs to ensure these can't be turned off, and turned on again after doing damage
    - Supported by most commercial database systems
    - But required DBAs with knowledge of application to monitor at this level
  - Keep archival copies and cross check periodically

Access Control in Application Layer

- Authorization in application layer vs. database layer
  - Benefits
    - fine grained authorizations, such as to individual tuples, can be implemented by the application.
    - authorizations based on business logic easier to code at application level
  - Drawback:
    - Authorization must be done in application code, and may be dispersed all over an application
    - Hard to check or modify authorizations
    - Checking for absence of authorization loopholes becomes very difficult since it requires reading large amounts of application code

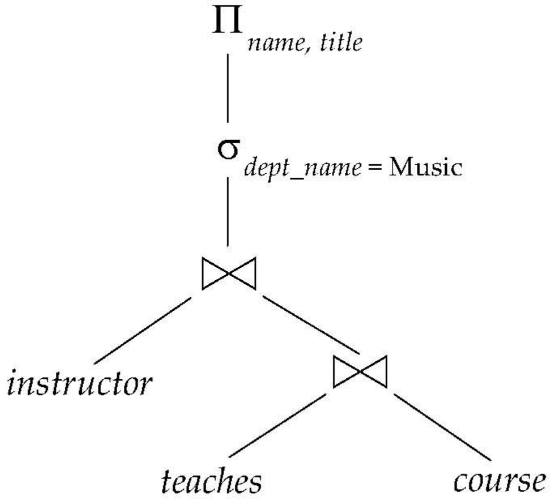
Introduction Query optimization

- **Query optimization**: finding the "best" **query execution plan** (**QEP**) among the many possible ones
  - User is not expected to write queries efficiently (DBMS optimizer takes care of that)
- Alternative ways to execute a given query – 2 levels
  - Equivalent relational algebra expressions
  - Different implementation choices for each relational algebra operation
    - Algorithms, indices, coordination between successive operations, …

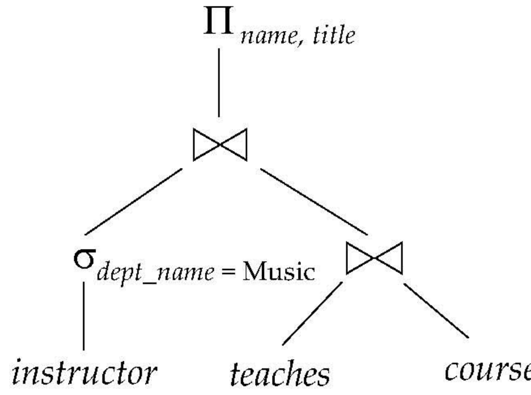
INSTR(i_id, name, dept_name, …)
COURSE(c_id, title, …)
TEACHES(i_id, c_id, …)

SELECT I.name, C.title
FROM INSTR I, COURSE C, TEACHES T
WHERE I.i_id = T.i_id
 AND T.c_id = C.c_id
AND dept_name="Music"

The name of all instructors in the department of Music together with the titles of all courses they teach


$\Pi_{name,\ title}$
$\sigma_{dept\_name = \text{Music}}$
$\bowtie$
instructor
$\bowtie$
teaches
course

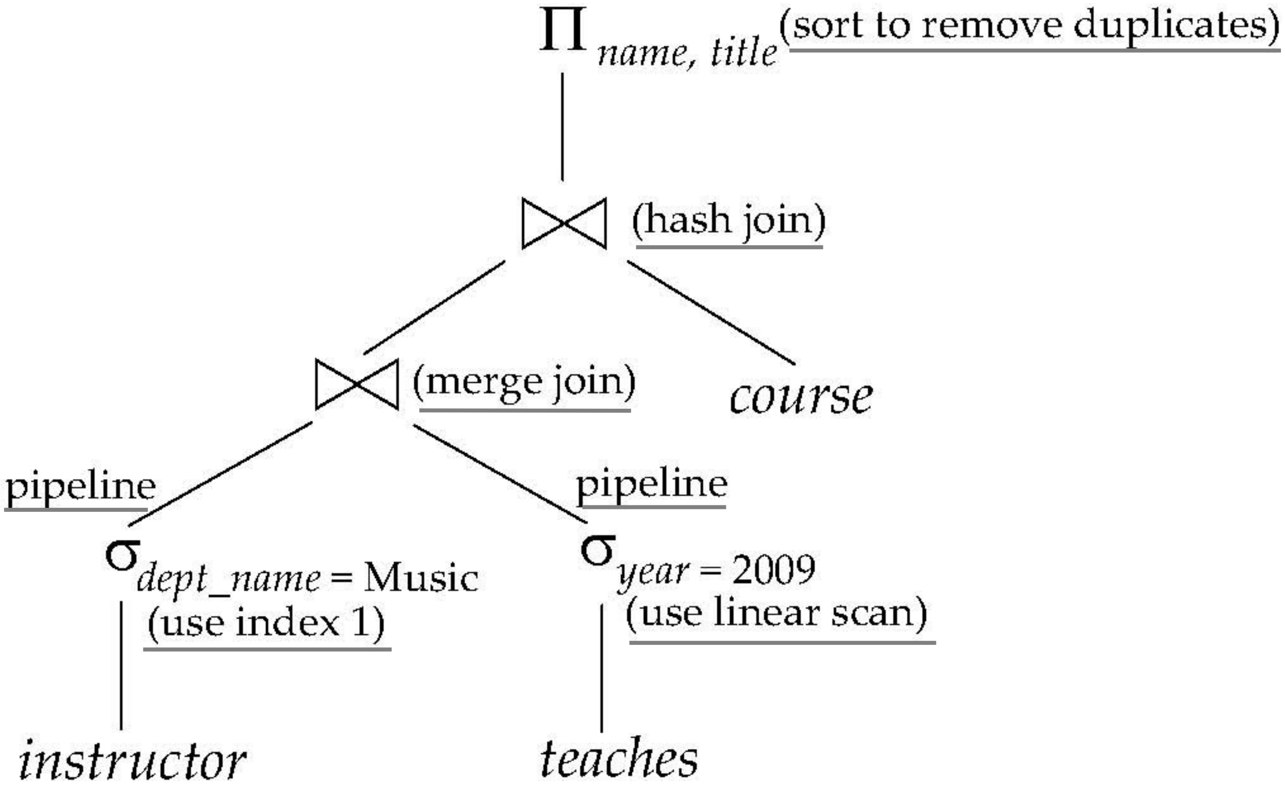
$\prod(\sigma(INSTR \bowtie (TEACHES \bowtie COURSE)))$


$\Pi_{name,\ title}$
$\bowtie$
$\sigma_{dept\_name = \text{Music}}$
$\bowtie$
instructor
teaches
cours

$\prod(\sigma(INSTR) \bowtie (TEACHES \bowtie COURSE))$

Introduction (Cont.)

- A **query evaluation plan** (**QEP**) defines exactly what algorithm is used for each operation, and how the execution of the operations is coordinated



$\Pi_{name,\ title}$
(sort to remove duplicates)
(hash join)
(merge join)
course
pipeline
pipeline
$\sigma_{dept\_name = \text{Music}}$
(use index 1)
$\sigma_{year = 2009}$
(use linear scan)
instructor
teaches

- Find out how to view query execution plans on your favorite database. For SQLite follow the link:

https://www.sqlite.org/eqp.html#:~:text=sqlite%3E%20CREATE%20INDEX%20i2%20ON,COVERING%20INDEX%20i2%20(a%3D%3F)&text=The%20order%20of%20the%20entries%20indicates%20the%20nesting%20order

Introduction (Cont.)

- Cost difference between query evaluation plans can be enormous
  - E.g. seconds vs. days in some cases
  - It is worth spending time in finding “best” QEP
- Steps in **cost-based query optimization**
  1. Generate logically equivalent expressions using **equivalence rules**
  2. Annotate in all possible ways resulting expressions to get alternative QEP
  3. Evaluate/estimate the cost (execution time) of each QEP
  4. Choose the cheapest QEP based on **estimated cost**
- Estimation of QEP cost based on:
  - Statistical information about relations (stored in the **Catalog**)
    - number of tuples, number of distinct values for an attribute
  - Statistics estimation for intermediate results
    - to compute cost of complex expressions
  - Cost formulae for algorithms, computed using statistics

Transformation of Relational Expressions

- Two relational algebra expressions are said to be **equivalent** if the two expressions generate the same set of tuples on every *legal* database instance
  - Note: order of tuples is irrelevant (and also order of attributes)
  - We don’t care if they generate different results on databases that violate integrity constraints (e.g., uniqueness of keys)
- In SQL, inputs and outputs are multisets (un-ordered) of tuples
  - Two expressions in the multiset version of the relational algebra are said to be equivalent if the two expressions generate the same multiset of tuples on every legal database instance
  - We focus on relational algebra and treat relations as sets
- An **equivalence rule** states that expressions of two forms are equivalent
  - One can replace an expression of first form by one of the second form, or vice versa

Equivalence Rules

1. Conjunctive selection operations can be deconstructed into a sequence of individual selections.

$$\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$$

2. Selection operations are commutative.

$$\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$$

3. Only the last in a sequence of projection operations is needed, the others can be omitted

$$\Pi_{L_1}(\Pi_{L_2}(\ldots(\Pi_{L_n}(E))\ldots)) = \Pi_{L_1}(E)$$

where $L_1 \subseteq L_2 \subseteq \ldots \subseteq L_n$

4. Selections can be combined with Cartesian products and theta joins.
   a. $\sigma_\theta(E_1 \text{ X } E_2) = E_1 \bowtie_\theta E_2$
   b. $\sigma_{\theta 1}(E_1 \bowtie_{\theta 2} E_2) = E_1 \bowtie_{\theta 1 \wedge \theta 2} E_2$

Equivalence Rules (Cont.)

5. Theta-join (and thus natural joins) operations are commutative.

$$E_1 \bowtie_\theta E_2 = E_2 \bowtie_\theta E_1$$

(but the order is important for efficiency)

6. (a) Natural join operations are associative:

$$(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$$

(again, the order is important for efficiency)

(b) Theta joins are associative in the following manner:

$$(E_1 \bowtie_{\theta_1} E_2) \bowtie_{\theta_2 \wedge \theta_3} E_3 = E_1 \bowtie_{\theta_1 \wedge \theta_3} (E_2 \bowtie_{\theta_2} E_3)$$

where θ_1 involves attributes from only E_1 and E_2

and θ_2 involves attributes from only E_2 and E_3

Equivalence Rules (Cont.)

7. (a) Selection distribute over theta join in the following manner:

$$\sigma_{\theta_1}(E_1 \bowtie_\theta E_2) = (\sigma_{\theta_1}(E_1)) \bowtie_\theta E_2$$

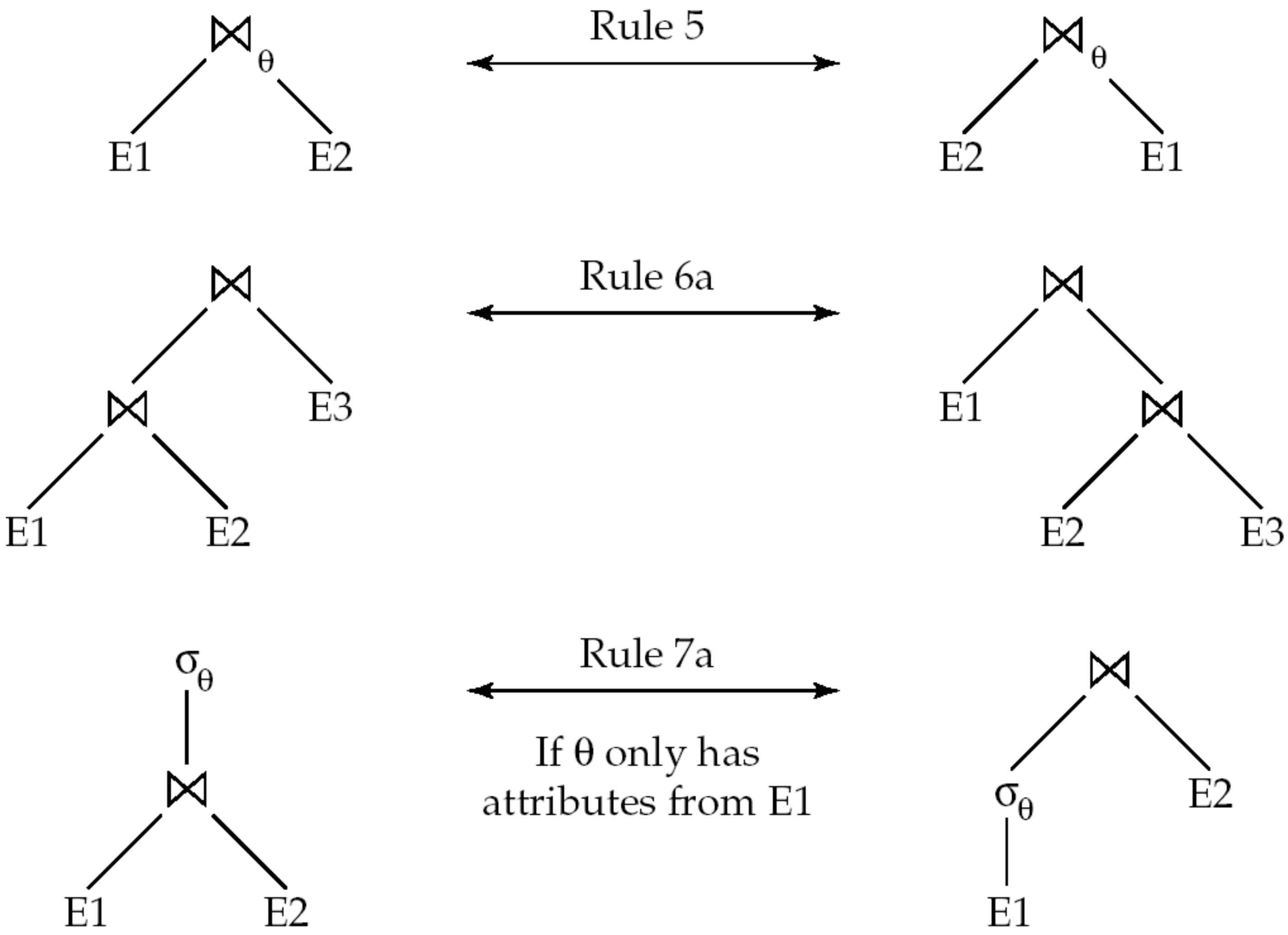
where θ_1 involves attributes from only E_1

(b) Complex selection distribute over theta join in the following manner:

$$\sigma_{\theta_1 \wedge \theta_2}(E_1 \bowtie_\theta E_2) = (\sigma_{\theta_1}(E_1)) \bowtie_\theta (\sigma_{\theta_2}(E_2))$$

where θ_1 involves attributes from only E_1
and θ_2 involves attributes from only E_2

Pictorial Depiction of Equivalence Rules


$\bowtie_\theta$
E1
E2
Rule 5
$\bowtie_\theta$
E2
E1
$\bowtie$
$\bowtie$
E3
E1
E2
Rule 6a
$\bowtie$
E1
$\bowtie$
E2
E3
σ_θ
$\bowtie$
E1
E2
Rule 7a
If θ only has attributes from E1
$\bowtie$
σ_θ
E2
E1

Exercise

- Disprove the equivalence

$$(R \mathbin{⟕} S) \mathbin{⟕} T = R \mathbin{⟕} (S \mathbin{⟕} T)$$

Definition (left outer join): the result of a left outer join $T = R \mathbin{⟕} S$ is a super-set of the result of the join $T' = R \bowtie S$ in that all tuples in T' appear in T. In addition, T preserve those tuples that are lost in the join, by creating tuples in T that are filled with *null* values

STUD	**stud_id**	**name**	**surname**
	1	gino	bianchi
	2	filippo	neri
	3	mario	rossi

TAKES	**stud_id**	**course**	**grade**
	1	Math	30
	2	DB	22
	2	Logic	30

$STUD \mathbin{⟕} TAKES$

stud_id	**name**	**surname**	**course**	**grade**
1	gino	bianchi	Math	30
2	filippo	neri	DB	22
2	filippo	neri	Logic	30
3	mario	rossi	***null***	***null***

$TAKES \mathbin{⟕} STUD$???

Solution

- Disprove the equivalence (R ⟕ S) ⟕ T = R ⟕ (S ⟕ T)

R

A	A_R
1	1

S

A	A_S
2	1

T

A	A_T
1	1

R ⟕ S

A	A_R	A_S
1	1	null

S ⟕ T

A	A_S	A_T
2	1	null

(R ⟕ S) ⟕ T

A	A_R	A_S	A_T
1	1	null	1

R ⟕ (S ⟕ T)

A	A_R	A_S	A_T
1	1	null	null

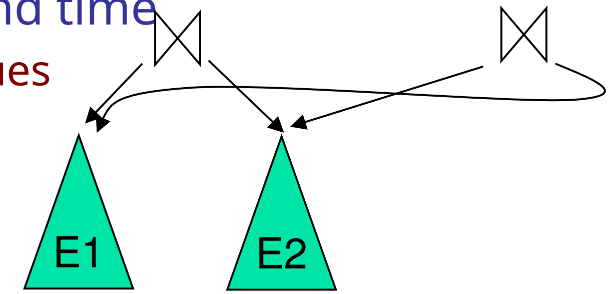
Equivalence derivability and minimality

- Some equivalence can be derived from others
  - example: 2 can be obtained from 1 (exploiting commutativity of conjunction)
  - 7b can be obtained from 1 and 7a
- Optimizers use **minimal** sets of equivalence rules

Enumeration of Equivalent Expressions

- Query optimizers use equivalence rules to **systematically** generate expressions equivalent to the given one
- Can generate all equivalent expressions as follows:
  - Repeat (starting from the set containing only the given expression)
    - apply all applicable equivalence rules on every sub-expression of every equivalent expression found so far
    - add newly generated expressions to the set of equivalent expressions

    Until no new equivalent expressions are generated
- The above approach is very expensive in space and time
  - Space: efficient expression-representation techniques
    - 1 copy is stored for shared sub-expressions
  - Time: partial generation
    - Dynamic programming
    - Greedy techniques (select best choices at each step)
    - Heuristics, e.g., single-relation operations (selections, projections) are pushed inside (performed earlier)



Cost Estimation

- Cost of each operator computed as described in Chapter 15 *
  - Need statistics of input relations
    - E.g. number of tuples, sizes of tuples
- Statistics are collected in the **Catalog**
- Inputs can be results of sub-expressions
  - Need to estimate statistics of expression results
    - Based on values taken by the attributes
  - Estimation of size of intermediate results
    - Based on # of tuple in input to successive operations
  - Estimation of number of distinct values in intermediate results
    - selectivity rate of successive selection operations i.e. # of select operations
- Statistics are not totally accurate
  - Information in the catalog might be not always up-to-date (delay)
  - A precise estimate for intermediate results might be impossible to compute

---

* **Silberschatz, Korth, and Sudarshan, *Database System Concepts*, 7° ed.**

Statistical Information for Cost Estimation

- Statistics information is maintained in the **Catalog**
- The catalog is itself stored in the database as relation(s)
- It contains:
  - n_r: number of tuples in a relation r
  - b_r: number of blocks containing tuples of r
  - l_r: size of a tuple of r (in bytes)
  - f_r: blocking factor of r – i.e., the number of tuples of r that fit into one block
  - $V(A, r)$: number of distinct values that appear in r for set of attributes A
    - $V(A, r) = \prod_A(r)$ – if A is a key, then $V(A,r) = n_r$
  - $min(A,r)$: smallest value appearing in relation r for set of attribute A;
  - $max(A,r)$: largest value appearing in relation r for set of attribute A;
  - statistics about indices (height of B^+-trees, number of blocks for leaves, …)
- We assume tuples of r are stored together physically in a file; then: $b_r = \lceil n_r / f_r \rceil$
- Information not always up-to-date
  - Catalog is not updated to every DB change (done during periods of light system load)

Selection Size Estimation

- # of records that will satisfy the selection predicate (aka selection condition)
- $\sigma_{A=V}(r)$ *(we are assuming that v actually is present in A)*
    - $n_r / V(A,r)$
    - 1 if A is key
- $\sigma_{A \leq V}(r)$ *(case $\sigma_{A \geq V}(r)$ is symmetric)*
    - 0 if $v < \min(A,r)$
    - n_r if $v >= \max(A,r)$
    - $n_r * \dfrac{v - \min(A,r)}{\max(A,r) - \min(A,r)}$ otherwise
    - In absence of statistical information or when *v* is unknown at time of cost estimation (e.g., *v* is computed at run-time by the application using the DB), then we assume
        - $n_r / 2$
- If histograms are available, we can do more precise estimates
    - use values for restricted ranges instead of n_r, $V(A,r)$, $\min(A, r)$, $\max(A,r)$
- A histogram is a graph that shows the frequency of numerical data using rectangles. The height of a rectangle (the vertical axis) represents the distribution frequency of a variable (the amount, or how often that variable appears). The width of the rectangle (horizontal axis) represents the value of the variable (for instance, minutes, years, or ages). Histograms make cost estimates accurately.

Complex Selection Size Estimation

- Conjunction $E = \sigma_{\theta_1 \wedge \theta_2 \wedge \ldots \wedge \theta_n}(r)$
  - we compute s_i = *size selection for θ_i $(i = 1,\ldots, n)$
  - **selectivity rate** (**SR****) of $\sigma_{\theta_i}(r)$: $SR(\sigma_{\theta_i}(r)) = s_i / n_r$ $(i = 1,\ldots, n)$
  - $SR(E) = \Pi_i (SR(\sigma_{\theta_i}(r))) = s_1 / n_r * \ldots * s_n / n_r$ Π_i is multiplication with $i = 1,\ldots,n$
  - # of record for $E = n_r * SR(E) =$ $\boxed{n_r * \dfrac{s_1 * s_2 * \ldots * s_n}{(n_r)^n}}$
- Disjunction $E = \sigma_{\theta_1 \vee \theta_2 \vee \ldots \vee \theta_n}(r) = \sigma_{\neg(\neg\theta_1 \wedge \neg\theta_2 \wedge \ldots \wedge \neg\theta_n)}(r)$ [
  - $SR(E) = 1 - SR(\sigma_{\neg\theta_1 \wedge \neg\theta_2 \wedge \ldots \wedge \neg\theta_n}(r))$
  - $SR(\sigma_{\neg\theta_1 \wedge \neg\theta_2 \wedge \ldots \wedge \neg\theta_n}(r)) = (1 - s_1 / n_r) * \ldots * (1 - s_n / n_r)$
  - # of record for $E = n_r * SR(E) =$ $\boxed{n_r * \left[1 - (1 - \dfrac{s_1}{n_r}) * (1 - \dfrac{s_2}{n_r}) * \ldots * (1 - \dfrac{s_n}{n_r})\right]}$
- Negation $E = \sigma_{\neg\theta}(r)$
  - # of record for $E = n_r$ - # of record for $\sigma_\theta(r)$

*This is the operation of selecting particular column/s from a table. Hence its size estimate **is equal to number of distinct values of column/s present** . **The selectivity is the fraction of rows in a table or partition that is chosen by the predicate.

Join Size Estimation

- # of records that will be included in the result
- *(cartesian product) r* x *s:* *# of records =* $n_r * n_s$
- *(natural join on attribute A) r* ⋈ *s:*
  - for each tuple t_r of *r* there are in average $n_s / V(A,s)$ many tuples of *s* selected
  - thus, **# of records =** $n_r * n_s / V(A,s)$
  - by switching the role of *r* and *s* we get **# of records =** $n_r * n_s / V(A,r)$
  - lowest is more accurate estimation *# of records =* $n_r * n_s / max\{ V(A,r), V(A,s) \}$
  - histograms can be used for more accurate estimations
    - histograms must be on join attributes, for both relations, and with same ranges
    - use values for restricted ranges instead of n_r , n_s , $V(A,r)$, $V(A,s)$ and then sum estimations for each range
  - if *A* is key for *r*, then *# of records <=* n_s (and vice versa)
    - in addition, if *A* is not null in *s*, then *# of records =* n_s (and vice versa)
- *(theta join)* $r \bowtie_\theta s$
  - $r \bowtie_\theta s = \sigma_\theta(r \times s)$ use formulas for cartesian product and selection

Size Estimation for Other Operations

- projection (no duplications): *# of records = V(A,r)*
- aggregation ${}_{G}\gamma_{F}(r)$ *# of records = V(G,r)*
- set operations
  - between selections on same relation use formulas for selection
    - es.: $\sigma_{\theta_1}(r) \cup \sigma_{\theta_2}(r) = \sigma_{\theta_1 \vee \theta_2}(r)$
  - r ∪ s *# of records =* $n_r + n_s$
  - r ∩ s *# of records = min* $\{ n_r , n_s \}$
  - r – s *# of records =* n_r
- outer join
  - left outer join *# of records = # of records for inner join* $+ n_r$
  - right outer join *# of records = # of records for inner join* $+ n_s$
  - full outer join *# of records = # of records for inner join* $+ n_r + n_s$

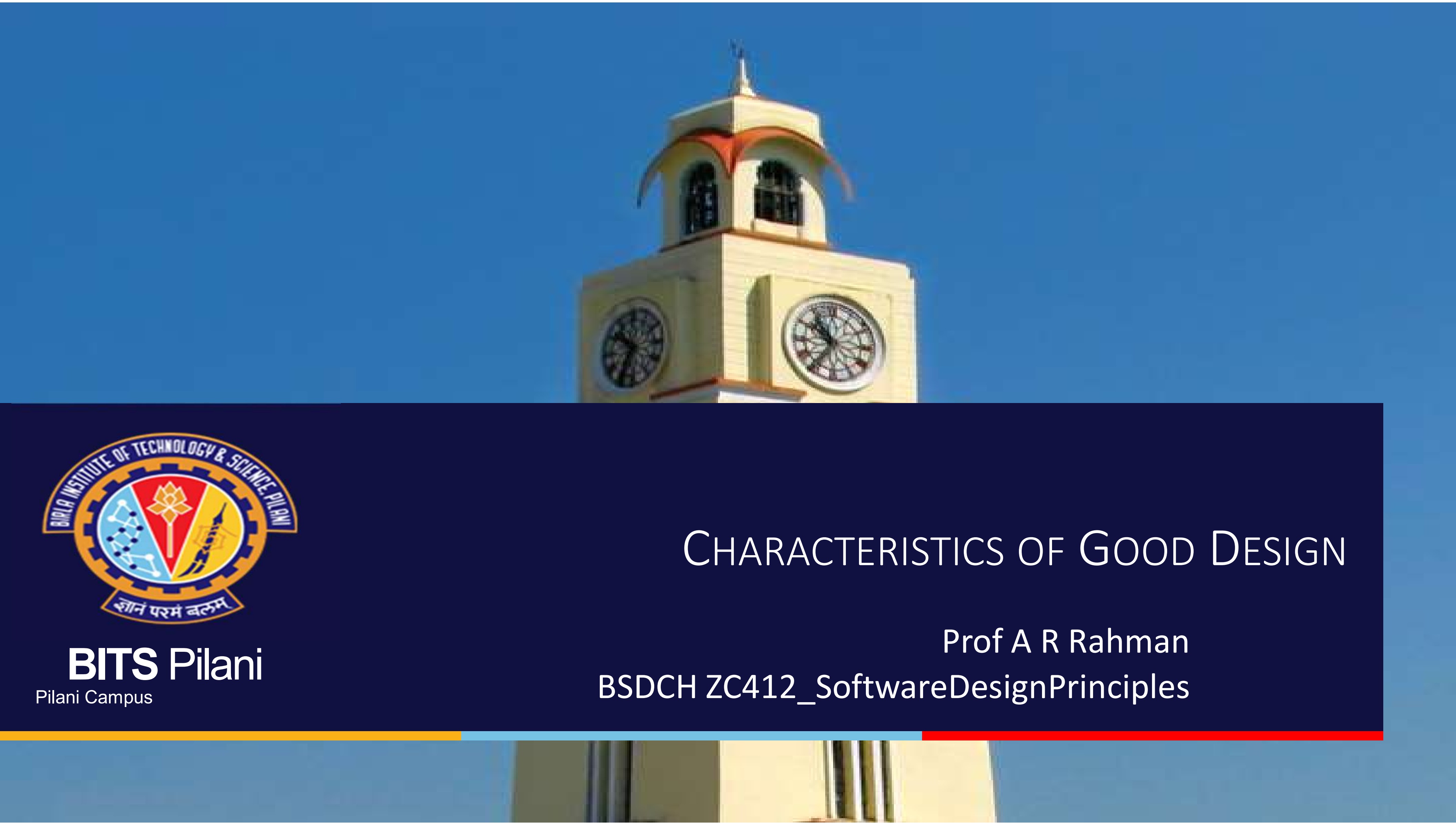
Estimation for Number of Distinct Values

- # distinct values in the result for expression E and attribute (or set of attributes) A: **$V(A,E)$**
- Selection $E = \sigma_\theta(r)$
  - $V(A, E)$ is a specific value for some conditions – e.g., $A=3$ or $3 < A <= 6$
  - condition $A < v$ (or $A > v$, $A >= v$, …) $V(A,E) = V(A,r)$ * *selectivity rate of the selection*
  - otherwise $V(A,E) = \min \{ n_E , V(A,r) \}$
- Join $E = r \bowtie s$
  - A only contains attributes from r $V(A,E) = \min \{ n_E , V(A,r) \}$
  - A only contains attributes from s $V(A,E) = \min \{ n_E , V(A,s) \}$
  - A contains attributes $A1$ from r and attributes $A2$ from s

    $V(A,E) = \min \{ n_E , V(A1, r) * V(A2 – A1, s) , V(A2, s) * V(A1 – A2, r) \}$

Choice of Evaluation Plans

- Must consider the interaction of evaluation techniques when choosing evaluation plans
  - choosing the cheapest algorithm for each operation independently may not yield best overall algorithm. E.g.
    - merge-join may be costlier than hash-join, but may provide a sorted output which reduces the cost for an outer level aggregation
    - nested-loop join may provide opportunity for pipelining
- Practical query optimizers incorporate elements of the following two broad approaches:
  1. Search all the plans and choose the best plan in a cost-based fashion
  2. Uses heuristics (shortcuts) to choose a plan



BITS Pilani
Pilani Campus

CHARACTERISTICS OF GOOD DESIGN

Prof A R Rahman
BSDCH ZC412_SoftwareDesignPrinciples

Characteristics of good design

CHARACTERISTICS OF GOOD SOFTWARE

- While developing any kind of software product.
- The first question in any developer's mind is, “What are the qualities that good software should have?“
- Well before going into technical characteristics, Let us see the state the obvious expectations one has from any software.
- A software product must meet all the requirements of the customer or end-user.
- Also, the cost of developing and maintaining the software should be low.
- The development of software should be completed in the specified time-frame.

CHARACTERISTICS OF GOOD SOFTWARE

- The obvious things which are expected from any project (and software development is a project in itself).

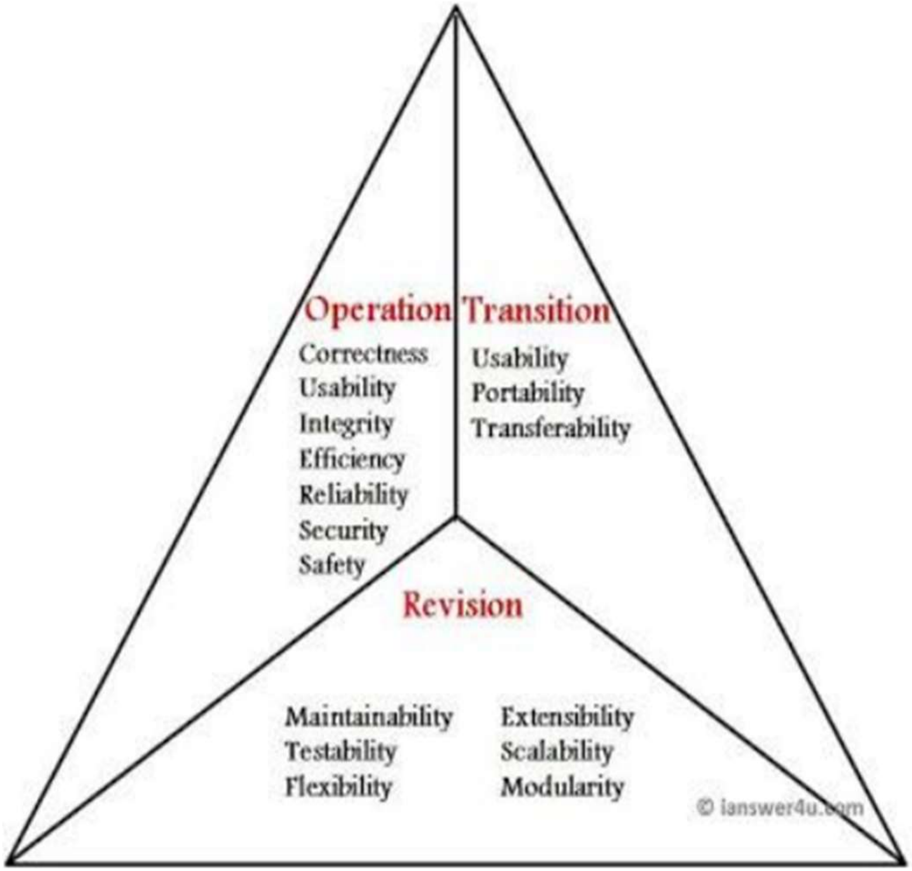
- Now lets take a look at Software Quality factors.

- These set of factors can be easily explained by Software Quality Triangle.

CHARACTERISTICS OF GOOD SOFTWARE

- The three characteristics of good application software are :-
- 1) Operational Characteristics
- 2) Transition Characteristics
- 3) Revision Characteristics

Software Quality Triangle


Operation
Correctness
Usability
Integrity
Efficiency
Reliability
Security
Safety
Transition
Usability
Portability
Transferability
Revision
Maintainability
Testability
Flexibility
Extensibility
Scalability
Modularity
© ianswer4u.com

Software Quality Triangle with characteristics

16 Characteristics of a Good Software

- What Operational Characteristics should a software have ?
- These are functionality based factors and related to 'exterior quality' of software. Various Operational Characteristics of software are :
- a) Correctness: The software which we are making should meet all the specifications stated by the customer.
- b) Usability/Learnability: The amount of efforts or time required to learn how to use the software should be less. This makes the software user-friendly even for IT-illiterate people

16 Characteristics of a Good Software

- c)Integrity : Just like medicines have side-effects, in the same way a software may have a side-effect i.e. it may affect the working of another application. But a quality software should not have side effects.

- d) Reliability : The software product should not have any defects. Not only this, it shouldn't fail while execution.

- e) Efficiency : This characteristic relates to the way software uses the available resources. The software should make effective use of the storage space and execute command as per desired timing requirements.

16 Characteristics of a Good Software

- f) Security : With the increase in security threats nowadays, this factor is gaining importance.
- The software shouldn't have ill effects on data / hardware.
- Proper measures should be taken to keep data secure from external threats.
- g) Safety : The software should not be hazardous to the environment/life.

What are the Revision Characteristics of software ?

- These engineering based factors of the relate to 'interior quality' of the software like efficiency, documentation and structure. These factors should be in-build in any good software.
- Various Revision Characteristics of software are :-
- a) Maintainability : Maintenance of the software should be easy for any kind of user.
- b) Flexibility : Changes in the software should be easy to make.
- c) Extensibility : It should be easy to increase the functions performed by it.
- d) Scalability : It should be very easy to upgrade it for more work(or for more number of users)

What are the Revision Characteristics of software ?

- e) Testability : Testing the software should be easy.

- f) Modularity : Any software is said to made of units and modules which are independent of each other.

- These modules are then integrated to make the final software.

- If the software is divided into separate independent parts that can be modified, tested separately, it has high modularity.

Transition Characteristics of the software :

- a) Interoperability : Interoperability is the ability of software to exchange information with other applications and make use of information transparently.

- b) Reusability : If we are able to use the software code with some modifications for different purpose then we call software to be reusable.

Transition Characteristics of the software :

- c)Portability : The ability of software to perform same functions across all environments and platforms, demonstrate its portability.

- Importance of any of these factors varies from application to application.

- In systems where human life is at stake, integrity and reliability factors must be given prime importance.

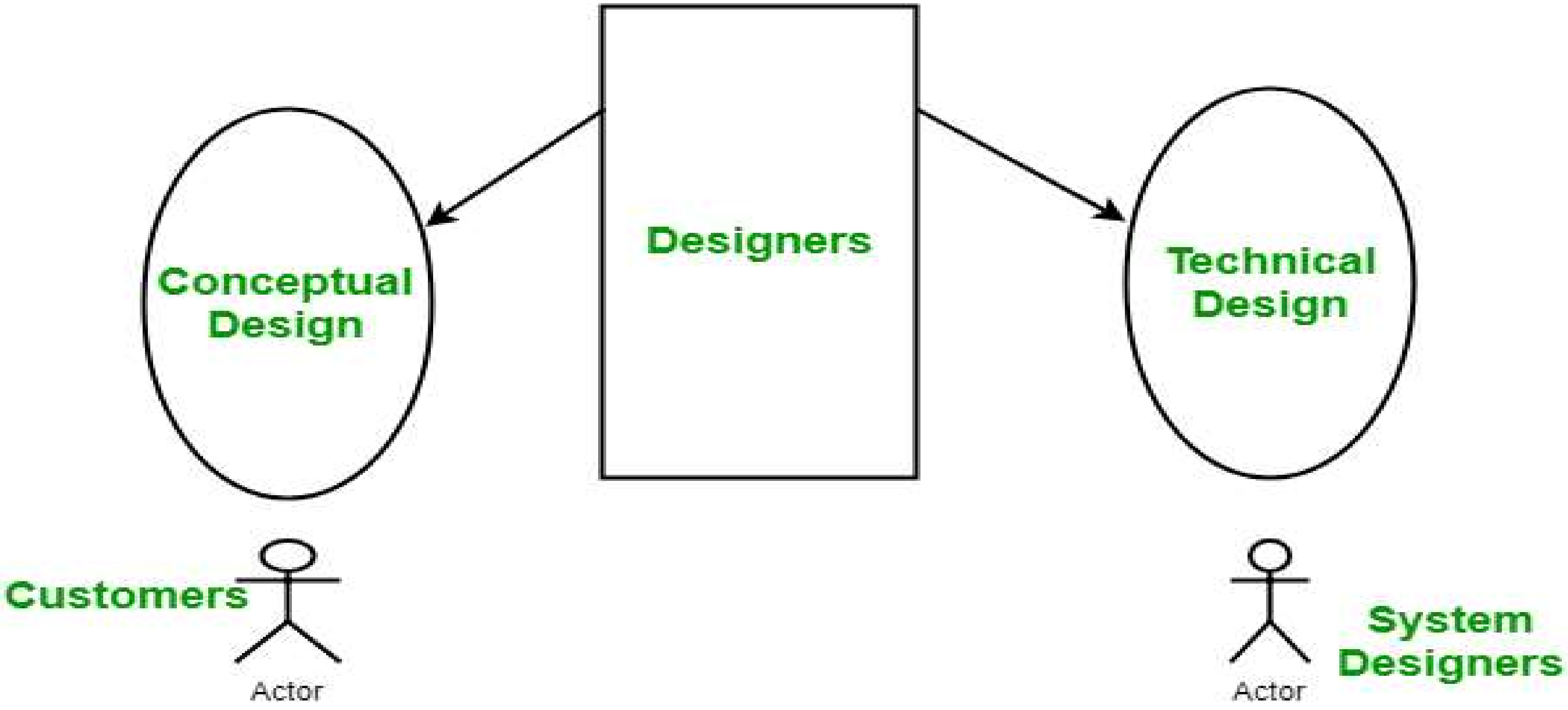
- In any business related application usability and maintainability are key factors to be considered.

- Always remember in Software Engineering, quality of software is everything, therefore try to deliver a product which has all these characteristics and qualities.

Characteristics of a good software design

- 1) Correctness.
- First of all, the design of any software is evaluated for its correctness.
- The evaluators check the software for every kind of input ...
- 2) Understandability.
- 3) Efficiency.
- 4) Maintainability.

Characteristics of a good software design


Designers
Conceptual Design
Technical Design
Customers
Actor
Actor
System Designers

Characteristics of good Software

- Software is treated as a good software by the means of different factors.

- A software product is concluded as a good software by what it offers and how well it can be used.

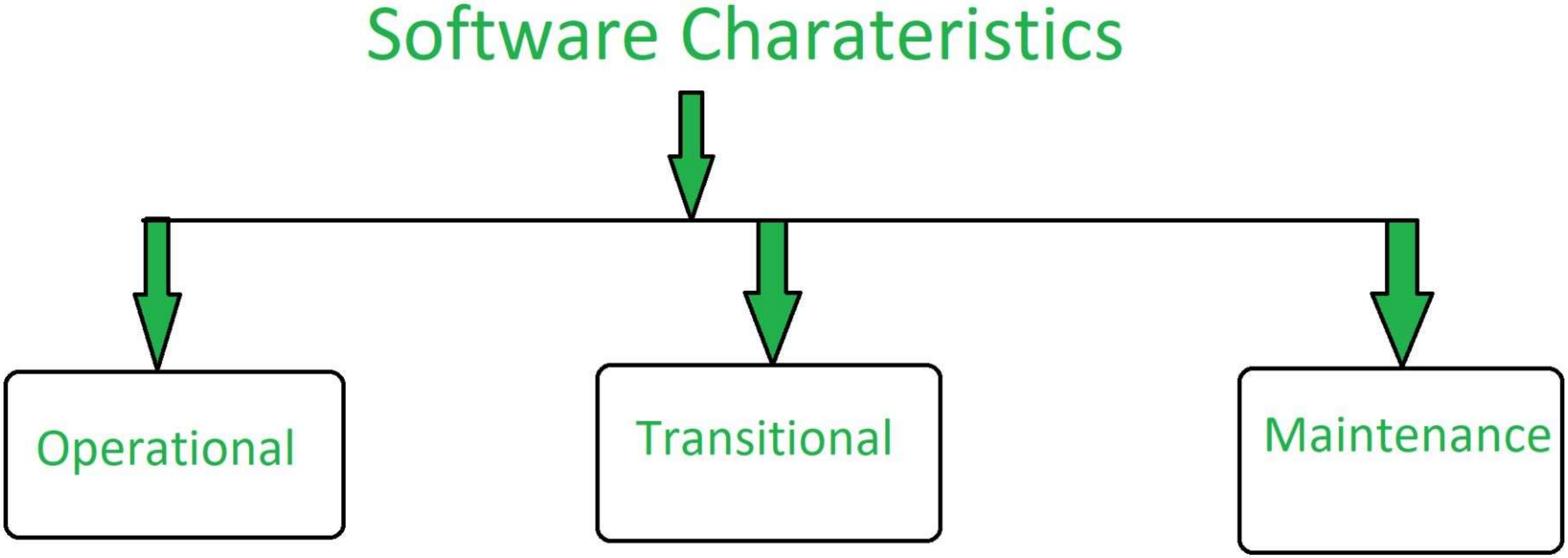
- The factors that decide the software properties are divided into three categories: Operational, Transitional, and Maintenance.

- These are explained as following below.

Characteristics of good Software

- 1. Operational:
  In operational categories, the factors that decide the software performance in operations. It can be measured on:
- Budget
- Usability
- Efficiency
- Correctness
- Functionality
- Dependability
- Security
- Safety

Characteristics of good Software



Software Charateristics

Operational

Transitional

Maintenance

Characteristics of good Software

- 2. Transitional:
  When the software is moved from one platform to another, the factors deciding the software quality:

- Portability

- Interoperability

- Reusability

- Adaptability

Characteristics of good Software

- 3. Maintenance:
  In this categories all factors are included that describes about how well a software has the capabilities to maintain itself in the ever changing environment:

- Modularity

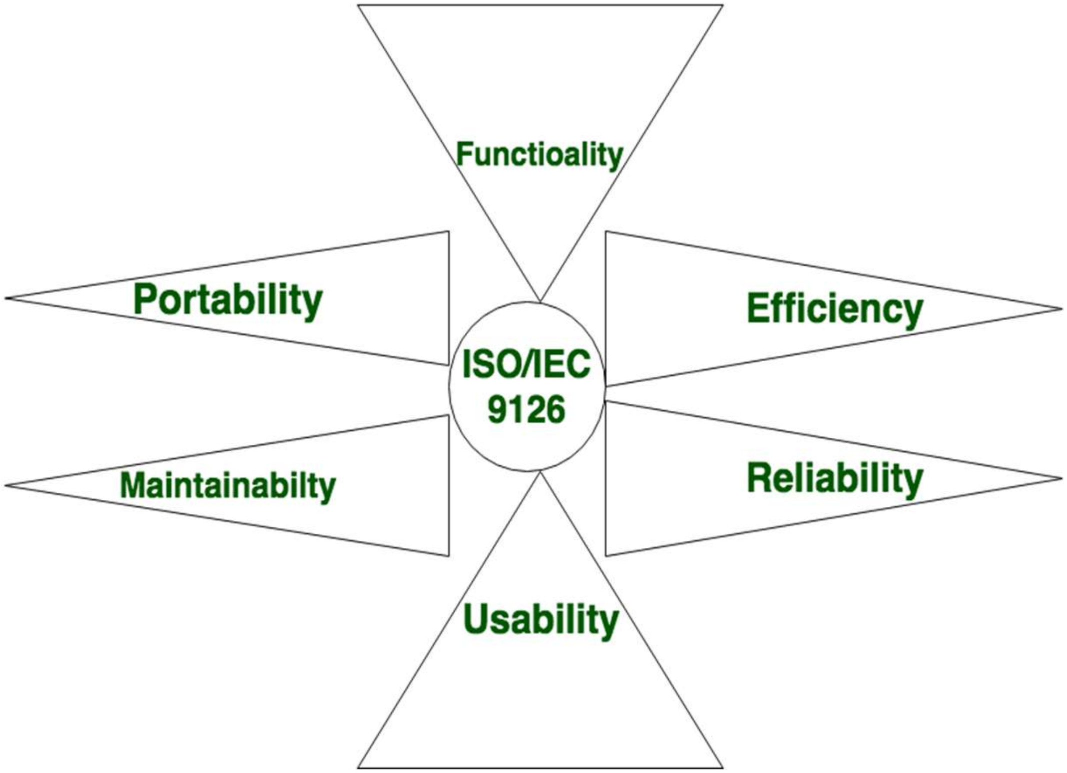
- Maintainability

- Flexibility

- Scalability

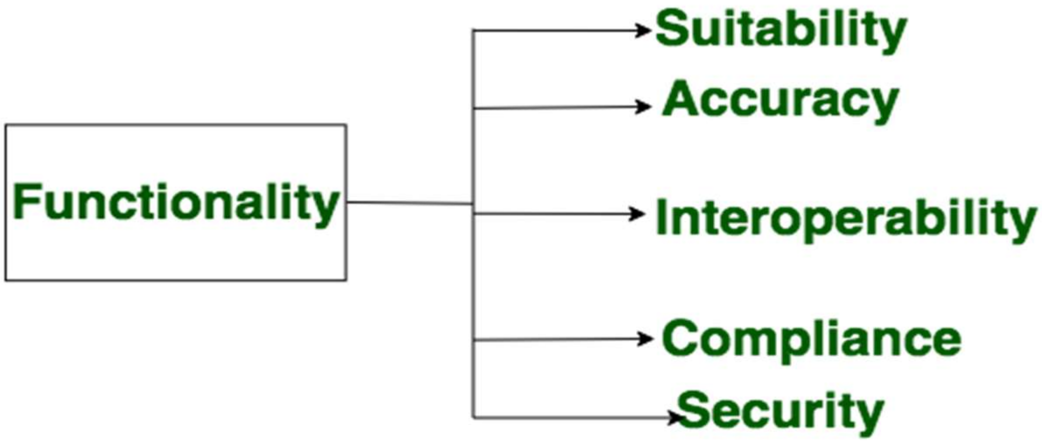
Software Characteristics

- **Software** is defined as a collection of computer programs, procedures, rules, and data.
- Software Characteristics are classified into six major components:


Functioality
Portability
Efficiency
ISO/IEC 9126
Maintainabilty
Reliability
Usability

Functionality:

- It refers to the degree of performance of the software against its intended purpose.
- Required functions are:



Functionality

Suitability

Accuracy

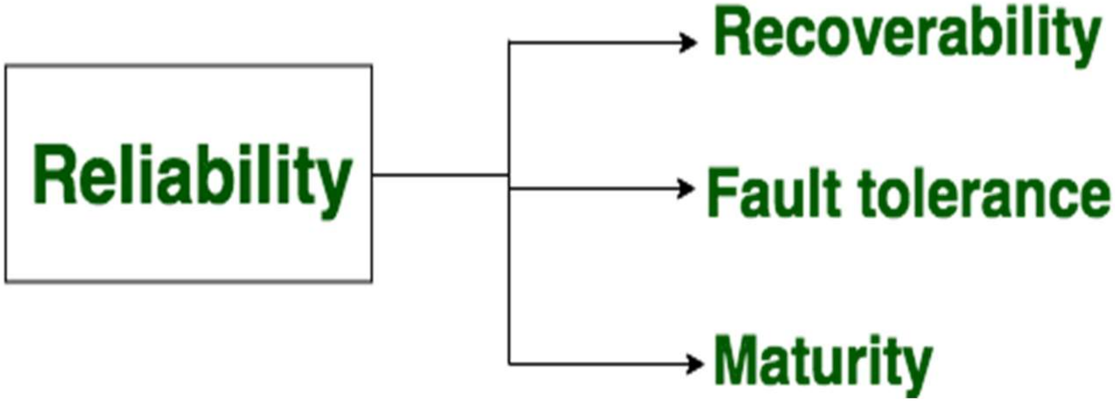
Interoperability

Compliance

Security

Reliability:

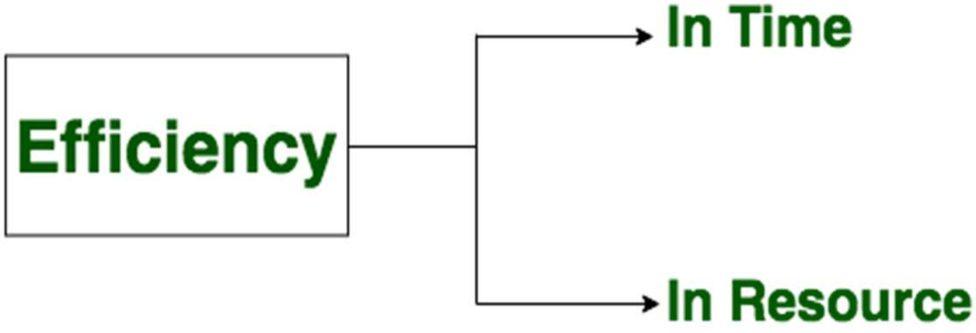
- A set of attributes that bears on the capability of software to maintain its level of performance under the given condition for a stated period of time. Required functions are:



Reliability
Recoverability
Fault tolerance
Maturity

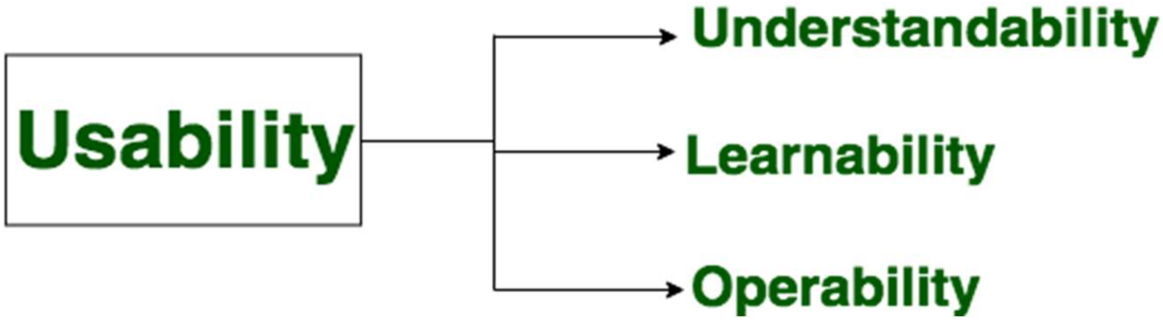
Efficiency:

- It refers to the ability of the software to use system resources in the most effective and efficient manner.
- The software should make effective use of storage space and executive command as per desired timing requirements.
- Required functions are:



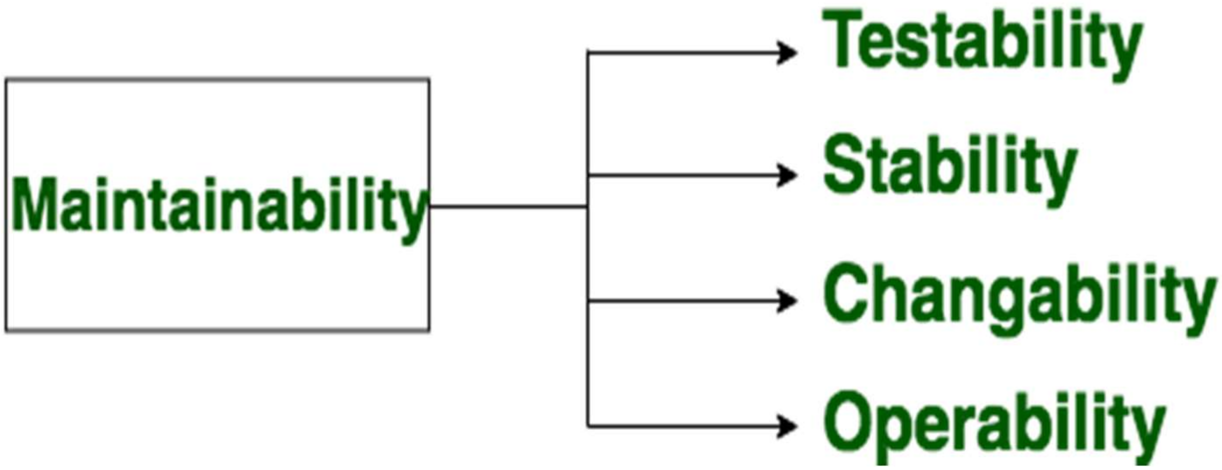
Usability:

- It refers to the extent to which the software can be used with ease.
- the amount of effort or time required to learn how to use the software.
- Required functions are:


Usability
Understandability
Learnability
Operability

Maintainability:

- It refers to the ease with which the modifications can be made in a software system to extend its functionality, improve its performance, or correct errors. Required functions are:



Maintainability

Testability

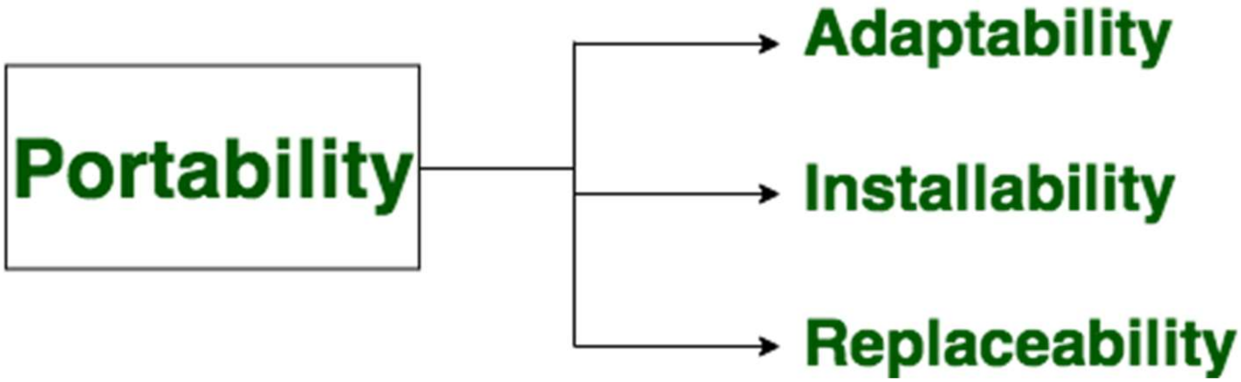
Stability

Changability

Operability

Portability:

- A set of attributes that bears on the ability of software to be transferred from one environment to another, without or minimum changes. Required functions are:



Portability

Adaptability

Installability

Replaceability

Software is developed or engineered; it is not manufactured in the classical sense:

- Although some similarities exist between software development and hardware manufacturing, few activities are fundamentally different.

- In both activities, high quality is achieved through good design, but the manufacturing phase for hardware can introduce quality problems than software.

The software doesn’t “wear out.”:

- Hardware components suffer from the growing effects of many other environmental factors. Stated simply, the hardware begins to wear out.

- Software is not susceptible to the environmental maladies that cause hardware to wear out.

- When a hardware component wears out, it is replaced by a spare part.

- There are no software spare parts.

The software doesn’t “wear out.”:

- Every software failure indicates an error in design or in the process through which design was translated into machine-executable code.

- Therefore, the software maintenance tasks that accommodate requests for change involve considerably more complexity than hardware maintenance.

- However, the implication is clear—the software doesn’t wear out.

- But it does deteriorate.

The software continues to be custom-built:

- A software part should be planned and carried out with the goal that it tends to be reused in various projects.

- Current reusable segments encapsulate the two information and the preparation that is applied to the information, empowering the programmer to make new applications from reusable parts.

- In the hardware world, component reuse is a natural part of the engineering process.

Grievances of A Software *a Sad Story of a Software...*

Hello! I am a software.

Yeh I know you use me every day, but do you even know who I am.

I am not Windows, Mac or Linux nor am I VLC, Edge, Mozilla or Chrome.

I am just like any other baby out there.

My developer, my parent, brought me into this world to help you with your various daily processes.

The lines of code, that he wrote, are my body parts.

And like you humans each of my parts has a function in itself.

After giving birth to me, my developer left me with his clients, to take care and use me according to their requirements.

Grievances of A Software *a Sad Story of a Software...*

You use me every day, still you don’t know me, nor do you understand my feelings.

I work for you day-in and day-out, still you expect more and more from me.

You use me to such an extent that I crash sometimes.

My body gets injured and needs time to recover.

Still you don’t understand my limitations.

Sometimes, some of my adopted parents upgrade me, give more functionality to my body parts.

Still your expectations never get fulfilled.

You throw me away after few years of use.

I have just a small appeal treat me as a *HUMAN*, because I have *FEELINGS* too.

Autobiography of a Word Processor

- Born in the year 1978, with tremendous efforts to create a unique word processor, I was yet another revolutionary product.

- They initially created me for the purpose of easing out their essay writing skills.

- As my newer versions came into place, I had become a software that could even tell mistakes, correct grammar and help with coding.

- With each version, I was brought in with something new and was not limited to just the functionality of a typewriter with modifying options.

- Providing perfection at every step and easing out typing had become my sole function.

Autobiography of a Word Processor

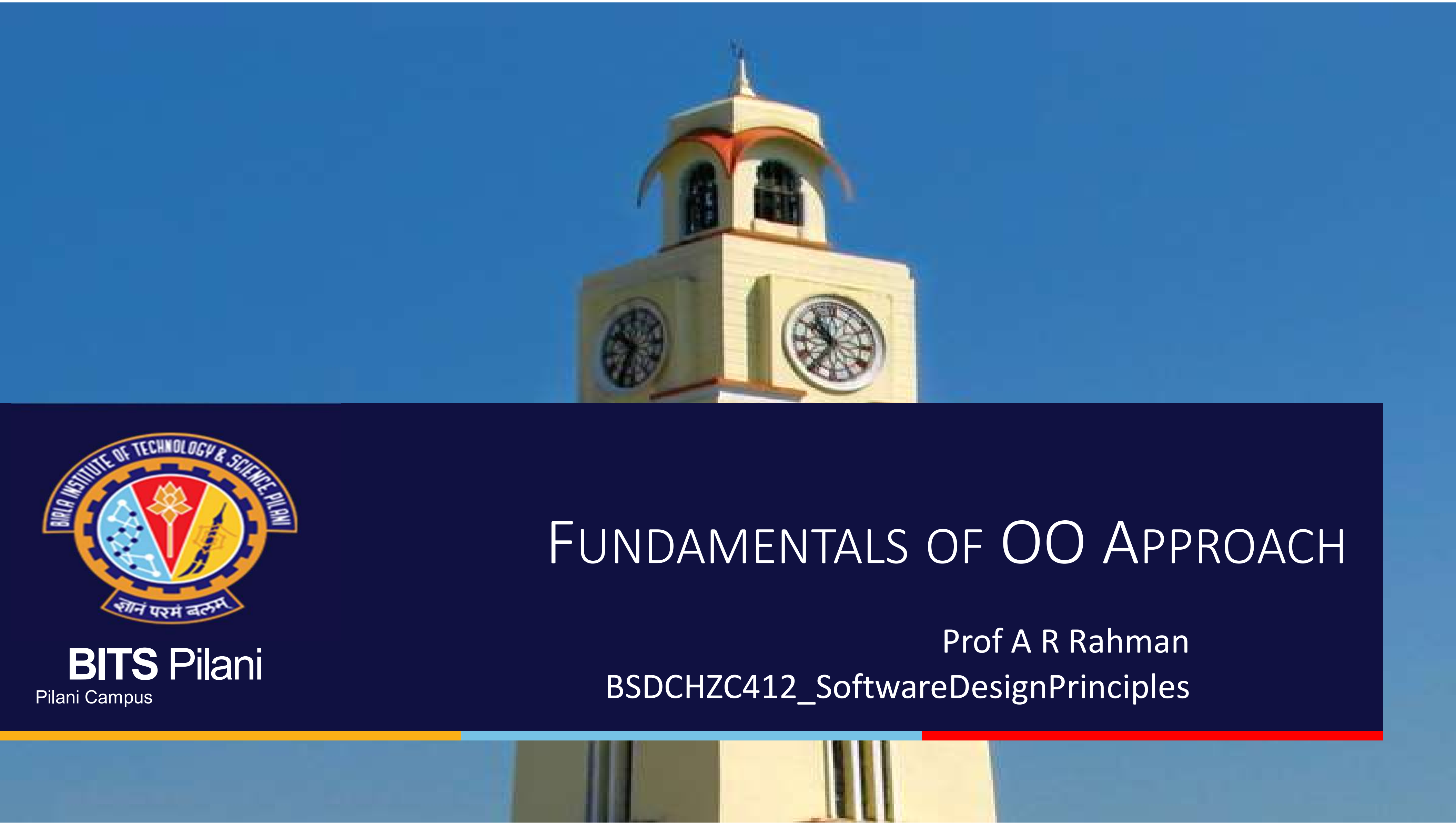
- Slowly and steadily, by 1986, I became the largest selling application software.
- But that is when my position started being threatened by other similar copied softwares developed by other companies.
- Since people always go in for better and advanced products, the story is the same for my counterparts too.
- They developed, they used, they left us.
- Harsh, but true. So they left me for the better ones by 1995.
- But soon, after the failure of others, I came into the limelight again by being used in the mobile revolution.

References- Design Flaws

- 10 Worst Design Failures of All Time – Don't Repeat These Fails (uxpin.com)
- 25 Serious Design Flaws In Everyday Products (list25.com)
- 10 exasperating design flaws in everyday objects | Creative Bloq
- 14 Types of Design Flaw – Simplicable
- 14 Overlooked Design Flaws In Everyday Things | Cracked.com

Object-Orientation Fundamental Concepts

Questions?



BITS Pilani
Pilani Campus

FUNDAMENTALS OF OO APPROACH

Prof A R Rahman
BSDCHZC412_SoftwareDesignPrinciples

Introduction to Object Oriented Development

Fundamentals of Object oriented approach

Object Oriented Programming

- Introduction to procedural programming techniques
- Modular programming techniques
- Object-oriented programming techniques
- Generic programming techniques

Object Oriented Programming

- Limitations of procedural programming

- Need for object-oriented programming

- Object Oriented Programming features

The Software Crisis

- We've all seen the statistics on software projects:
  - For every 6 new large-scale systems put into production, 2 are cancelled.
  - The average software project overshoots its schedule by 50%
    - Large projects generally do worse
  - 75% of all systems are "operating failures" that either do not function as intended or are not used at all.
- One of the primary reasons for the software crisis is that the complexity of the required systems has outstripped the complexity management capabilities of structured (procedurally based) software paradigms.

CONTENTS

- Objects
- Classes & Class as ADT
- Data members
- Methods
- Message Passing

CONTENTS

- Data encapsulation
- Data abstraction
- Information hiding
- Inheritance(Code Reuse)
- Polymorphism(Function overloading and operator overloading)

PROGRAMMING TECHNIQUES

- Unstructured programming
- Procedural programming
- Modular programming
- Object-oriented programming
- Generic programming

UNSTRUCTURED PROGRAMMING

- Only one Program i.e. main Program
- All the sequences of commands or statements in one programs called main Program
- E.g. Fortran, assembly language, old Basic

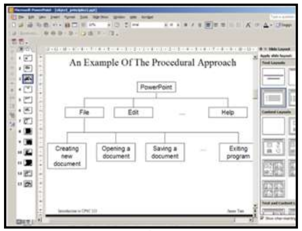
program
main program
data

Unstructured programming. The main program directly operates on global data.

The Procedural Approach (Presentation Software)

- Break down the program by what it does (described with *actions/verbs*)

PowerPoint

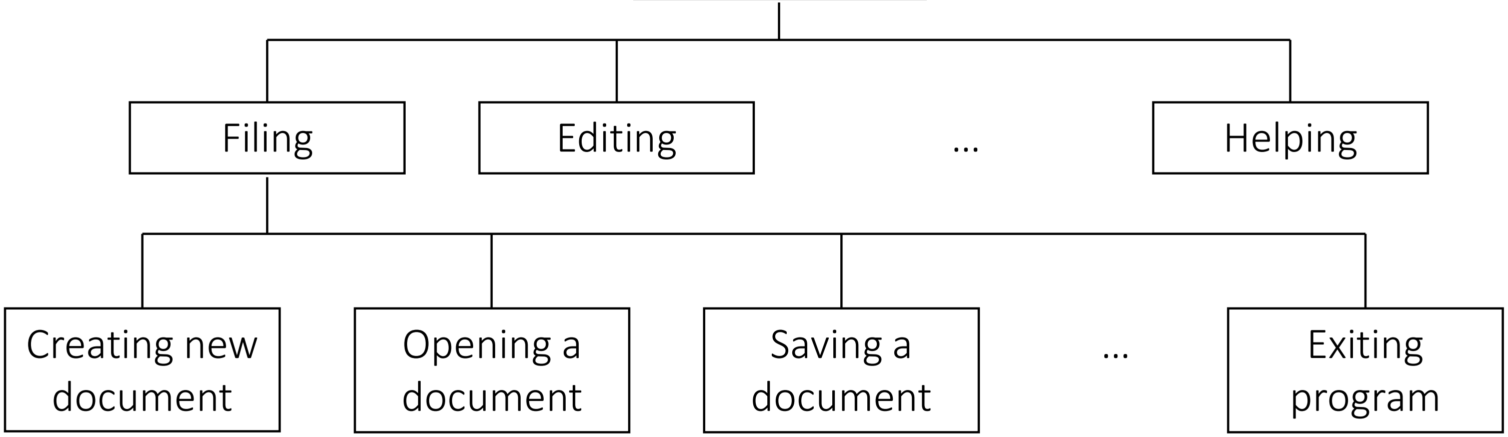


An Example Of The Procedural Approach

PowerPoint

File
Edit
...
Help

Creating new document
Opening a document
Saving a document
...
Exiting program



Filing
Editing
...
Helping

Creating new document
Opening a document
Saving a document
...
Exiting program

Procedural programming

Procedural programming is by far the most common form of programming. A program is a series of instructions which operate on variables. It is also known as imperative programming

Examples of procedural programming languages include FORTRAN, ALGOL, Pascal, C, MODULA2, Ada, BASIC. Despite their differences they all share the common characteristics of procedural programming

Advantages of procedural programming include its relative simplicity, and ease of implementation of compilers and interpreters.

Disadvantages of procedural programming include the difficulties of reasoning about programs and to some degree difficulty of parallelization.

Procedural programming tends to be relatively low level compared to some other paradigms, and as a result can be very much less productive.

STRUCTURED PROGRAMMING

- Also called as Procedural Programming
- For each task procedure is created
- The procedures are called from main

Figure 2.2: Execution of procedures. After processing flow of controls proceed where the call was made.

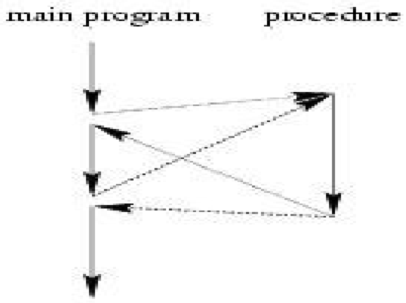
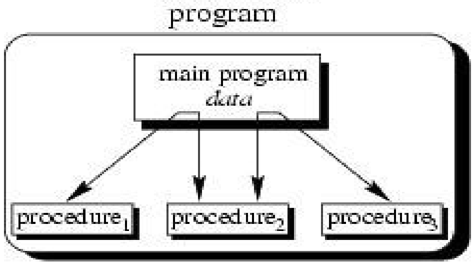

main program
procedure

Figure 2.3: Procedural programming. The main program coordinates calls to procedures and hands over appropriate data as parameters.


program
main program
data
procedure1
procedure2
procedure3

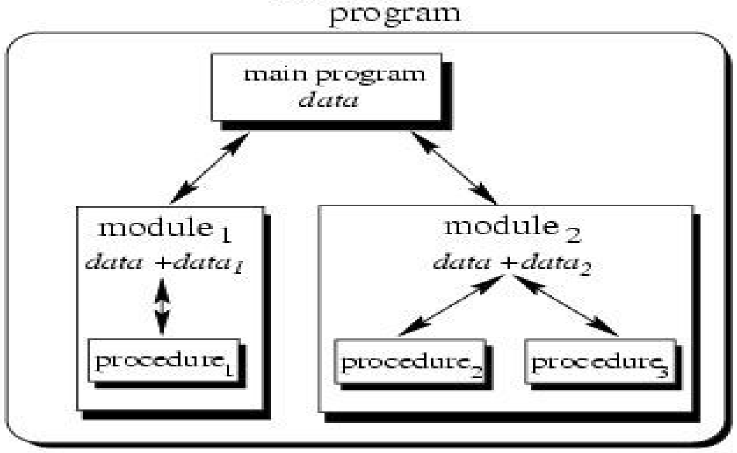
MODULAR PROGRAMMING

- Procedures with some common functionality are grouped together into separate modules

- Program is categorized into several smaller modules

- Each module can have its own data

Figure 2.4: Modular programming. The main program coordinates calls to procedures in separate modules and hands over appropriate data as parameters.


program
main program
data
module $_1$
data + *data* $_1$
procedure$_1$
module $_2$
data + *data* $_2$
procedure$_2$
procedure$_3$

The Object-Oriented Paradigm

- Because the O-O paradigm is very much about complexity, it has shown to be difficult to teach and learn
  - Almost all examples are too simple or contrived
    - They illustrate a point, but have no real meaning
    - Real examples are expensive to create and require a large time commitment to learn (due to complexity)
  - Students are often left with the feeling, "why would I do this?“
  - When confronted with time pressures, programmers will often fall back on what they know best (usually, procedural programming techniques)

The Object-Oriented Paradigm

- The average programmer takes 6 months to 2 years to learn the O-O paradigm.
  - There are, generally speaking, no short-cuts
    - Review your progress! Keep all the code you've written and review
  - Requires abstract thinking
  - Most students report a time when, "the light goes on“
    - The "a-ha" experience

Where it all starts...

- The focus of the Object-Oriented paradigm is the Object
  - NOT classes. Classes are an abstraction of objects.
- Objects are identified from the real world
- An object model is created to represent the real world
  - The data associated with each object are identified and modelled.
  - The various behaviours of each object are identified and modelled.
  - Interactions between objects are modelled.

Where it all starts...

- The Objects in the object model are classified
  - Designers create classes
  - Programmers implement classes
  - Classes are logically grouped into a hierarchical classification structure
    - Because Java only allows single inheritance, that structure is a tree.
    - C++ allows multiple inheritances; the classification structure is a lattice.

Object oriented programming

- Object oriented programming is characterized by the defining of classes of objects, and their properties.

- Inheritance of properties is one way of reducing the amount of programming, and provision of class libraries in the programming environment can also reduce the effort required.

- Features Of Object-Oriented Programming The most widely used object oriented language is C++ which provides object extensions to C, but this is rapidly being overtaken by Java.

Features Of Object-Oriented Programming

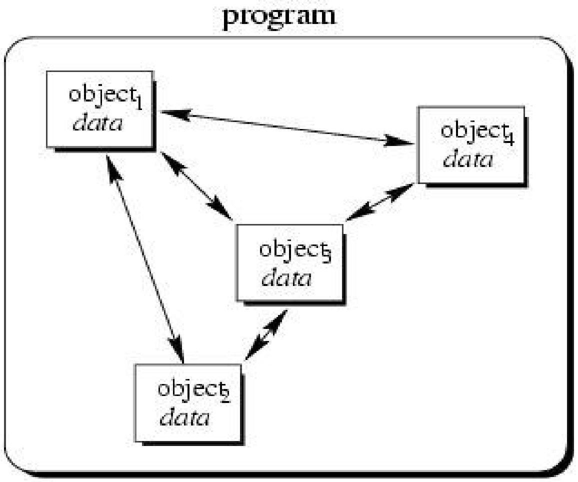
- Data Abstraction and Encapsulation
- Operations on the data are considered to be part of the data type,
- We can understand and use a data type without knowing all of its implementation details Neither how the data is represented nor how the operations are implemented
- We just need to know the interface (or method headers) – how to “communicate” with the object Compare to functional abstraction with methods

OBJECT-ORIENTED PROGRAMMING

- Works on objects which is considered smallest unit of the object oriented languages

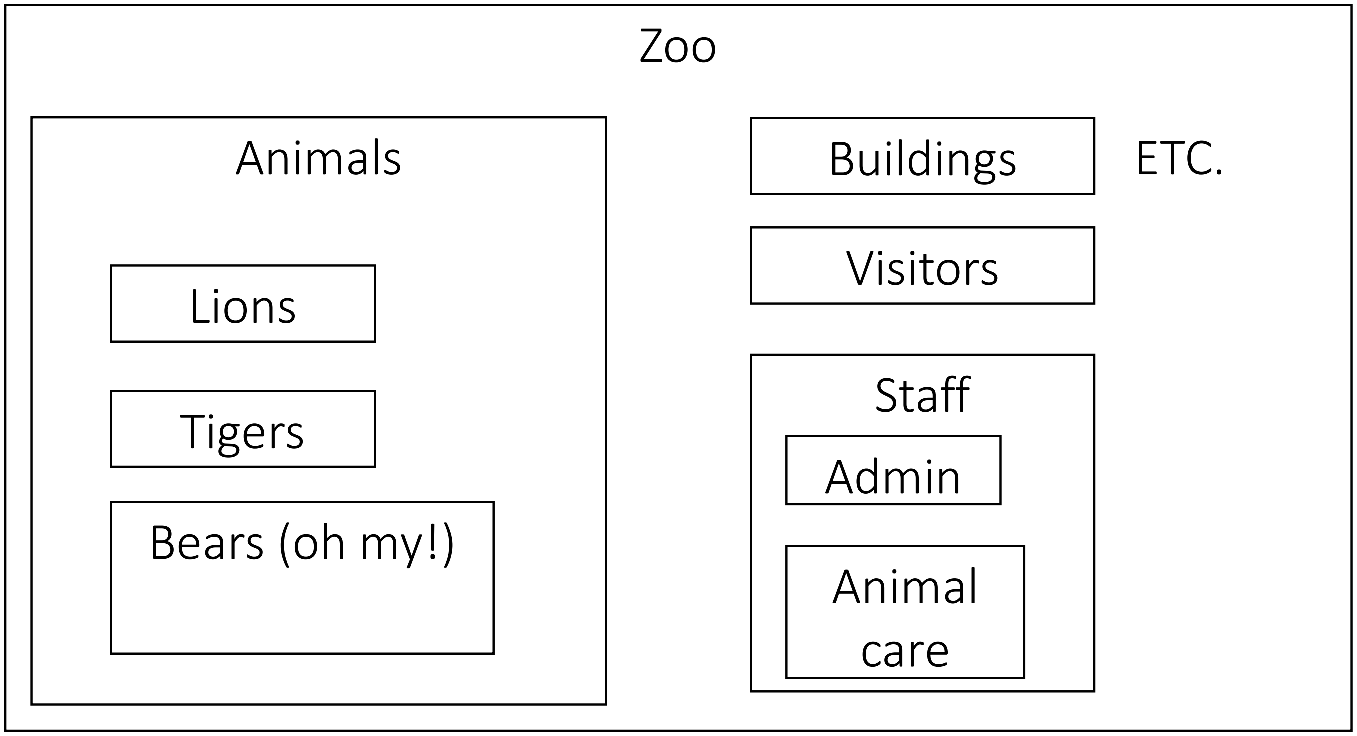
- Focuses more on data rather than Procedures

Figure 2.6: Object-oriented programming. Objects of the program interact by sending messages to each other.


program
object_1
data
object_4
data
object_3
data
object_2
data

An Example Of The Object-Oriented Approach (Simulation)

- Break down the program into entities (classes/objects - described with *nouns*)


Zoo
Animals
Lions
Tigers
Bears (oh my!)
Buildings
ETC.
Visitors
Staff
Admin
Animal care

EXAMPLE

- Unstructured Programming:

```
#include
void main()
{
 int a,b,c;
 clrscr();
 cout << "Enter the first number";  cin >> a;
 cout << "Enter the second number";  cin >> b;
 c=a+b;
 cout << "The sum is:" << c;  getch();
}
```

EXAMPLE

- Procedural Programming:

```
#include
int add(int, int);
void main()
{
  int a,b,c;
  clrscr();
  cout << "Enter the first number"; cin >> a;
  cout << "Enter the second number"; cin >> b;
  c=add(a,b);
  cout<<"The sum is:" << c;
  getch();
}
int add(int x, int y)
{
  int z=x+y;
  return z;
}
```

Object Oriented programming

```
#include
class Addition
{
  int a,b,c;  public:
  void read()
  {
    cin >> a;  cin >> b;
  }
  void add()
  {
    c=a+b;
  }
  void display()
  {
    cout << "The sum is:" << c;
  }
};
```

```
void main()
{
  Addition obj;   //object creation
 cout << "Enter the numbers";
 obj.read();
 obj.add();  obj.display();
 getch();      }
```

LIMITATIONS OF PROCEDURAL PROGRAMMING

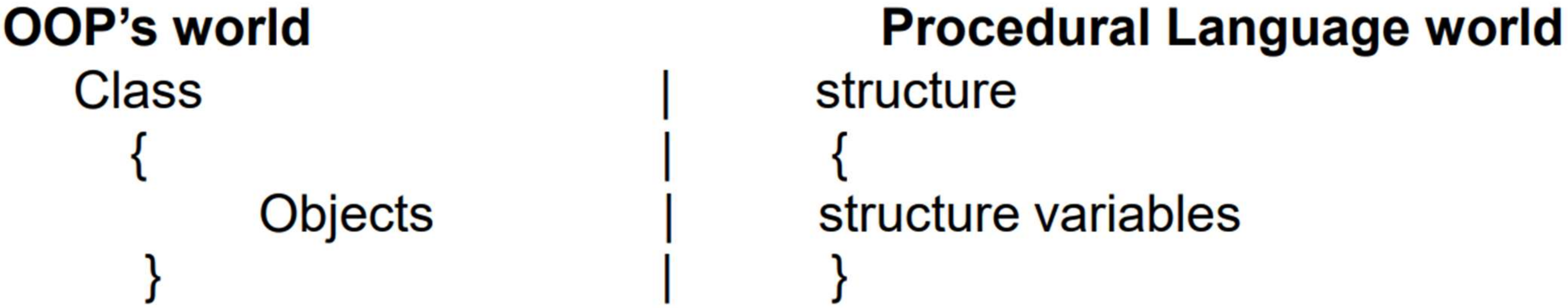
- Poor real world model.
- No importance to data.
- No privacy.
- No true reuse.
- Functions and data should be treated equally.

Feature	Procedure oriented Programming	Object oriented Programming
Divided Into	In POP Program is divided into small parts called **functions**	In OOP, program is divided into parts called **objects**.
Importance	In POP, Importance is not given to **data** but to functions as well as **sequence** of actions to be done.	In OOP, Importance is given to the data rather than procedures or functions because it works as a **real world**.
Approach	POP follows **Top Down approach**.	OOP follows **Bottom Up approach**
Access Specifiers	POP does not have any access specifier	OOP has access specifiers named Public, Private, Protected, etc.
Data Moving	In POP, Data can move freely from function to function in the system.	In OOP, objects can move and communicate with each other through member functions.
Expansion	To add new data and function in POP is not so easy.	OOP provides an easy way to add new data and function
Data Access	In POP, Most function uses Global data for sharing that can be accessed freely from function to function in the system.	In OOP, data can not move easily from or private so we can control the access of data
Data Hiding	POP does not have any proper way for hiding data so it is **less secure**.	OOP provides Data Hiding so provides **more security**.
Overloading	In POP, Overloading is not possible.	In OOP, overloading is possible in the form of Function Overloading and Operator Overloading.
Examples	Example of POP are : C, VB, FORTRAN, Pascal.	Example of OOP are : C++, JAVA, VB.NET, C#.NET.

GENERIC PROGRAMMING

- It is an idea that code should be as generic as possible
- Means writing templates
- e.g. container classes like arrays

OOP Vs Procedural languages



OOP's world	**Procedural Language world**
Class	structure
{	{
Objects	structure variables
}	}

OBJECT ORIENTED FEATURES

INTRODUCTION

- C++ is a statically typed, compiled, general-purpose, case-sensitive, free-form programming language that supports procedural, object-oriented, and generic programming.
- C++ is regarded as a middle-level language, as it comprises a combination of both high-level and low-level language features
- Developed by Bjarne Stroustrup starting in 1979 at Bell Labs in Murray Hill, New Jersey

CLASS DEFINITION

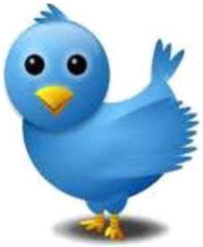
- It is template, format or blueprint
- Also can be said as user defined Data Type

- A class definition starts with the keyword class followed by the class name; and the class body enclosed by a pair of curly braces
- Class definition must be followed by a semicolon
- Class contains data members and member function

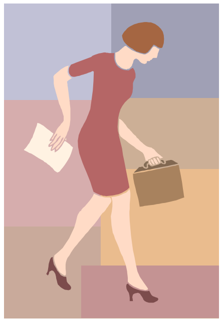
```
class Box{
     public:
     double length;           //     Length of a box
     double breadth;  //    Breadth of a box
     double height ;    //    Height of a box
  };
```

What Are Methods

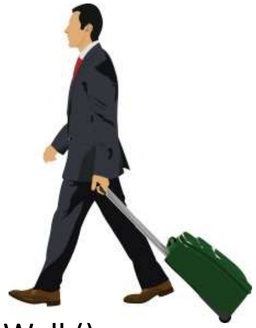
- Possible behaviors or actions for each instance (example) of a class.



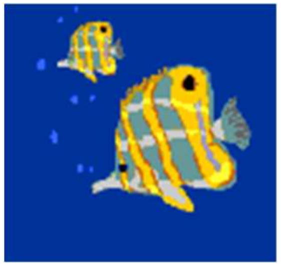
Fly()



Walk()
Talk()



Walk()
Talk()



Swim()

Instantiation

- New definition: Instantiation, creating a new instance or example of a class.
- Instances of a class are referred to as *objects*.
- Format:

  <class name> <instance name> = new *<class name>*()*;*

- Examples:

  ```
   Person ar = new Person();
  Scanner in = new Scanner(System.in);
  ```

Creates new object

Variable names: ‘ar’, ‘in’

Defining The Methods Of A Class In Java

Format:

<access modifier>[1] *<return type*[2]*> <method name>* (*<p1 type> <p1 name>*, *<p2 type> <p2 name>*…)
{
<Body of the method>
}

Example:

```
  public class Person
 {
   // Method definition
   public void sayAge() {
     System.out.println("My age is " + age);
   }
 }
```

1) For now *set the access modifier on all your methods to 'public'.*

2) Return types: includes all the built-in 'simple' types such as char, int, double…arrays and classes that have already been defined (as part of Java or third party extras)

Calling Methods: Outside The Class You’ve Defined

- Calling a method outside the body of the class (i.e., in another class definition)

- The method must be prefaced by a variable (actually a reference to an object – more on this later

```
public class Driver {
    public static void main(String [] args) {
        Person bart = new Person();
        Person lisa = new Person();
        // Incorrect! Who ages?
        becomeOlder();
        // Correct. Happy birthday Bart!
        bart.becomeOlder();
    }
}
```

Calling Methods: Inside The Class

- Calling a method inside the body of the class (where the method has been defined)
  - You can just directly refer to the method (or attribute)

```
public class Person {
  private int age;

  public void birthday() {
    becomeOlder();
  }

  public void becomeOlder() {
     age++;
  }
```

Class Person

```
public class Person
{    private int age;
    public Person()
    {
        Scanner in = new Scanner(System.in);
        System.out.print("Enter age: ");
        age = in.nextInt();
    }
    public void sayAge()
    {
        System.out.println("My age is " + age);
    }
}
```

Creating An Object

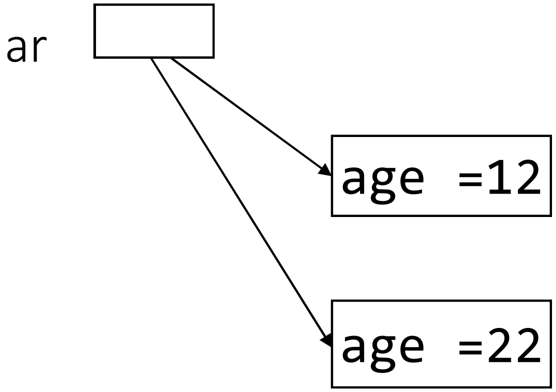
- Two stages (can be combined but don’t forget a step)
  - Create a variable that refers to an object e.g., Person ar;
  - Create a *new* object e.g., ar = new Person();
    - The keyword ‘new’ calls the constructor to create a new object in memory
  - Observe the following

Person ar;

ar = new Person(12);

ar = new Person(22);

ar is a reference to a Person object



ar

age =12

age =22

Terminology: Methods Vs. Functions

- Both include defining a block of code that be invoked via the name of the method or function (e.g., print())
- Methods a block of code that is *defined within a class definition* (Java example):

```
public class Person
{
   public Person() { ... }

   public void sayAge() { ... }
}
```

- Every object that is an instance of this class (e.g., ar is an instance of a Person) will be able to invoke these methods.

```
Person ar = new Person();
ar.sayAge();
```

Terminology: Methods Vs. Functions (2)

- **Functions** a block of code that is *defined outside or independent of a class* (Python example – it's largely not possible to do this in Java):
  - # Defining method sayBye()
  - class Person:
  -     def sayBye(self):
  -         print(“Open Sesame!")
  - # Method are called via an object
  - ar = Person()
  - ar.sayBye()
  - # Defining function: sayBye()
  - def sayBye():
  -     print(" Open Sesame!")

  - # Functions are called without creating an object
  - sayBye()

Methods Vs. Functions: Summary & Recap

Methods	Functions
• The Object-Oriented approach to program decomposition.	• The procedural (procedure = function) approach to program decomposition.
• Break the program down into classes.	• Break the program down into functions.
• Each class will have a number of methods.	• Functions can be invoked or called without creating any objects.
• Methods are invoked/called through an instance of a class (an object).	

Second Example: Second Look

Person.java

Calls in .java

```
Person ar = new Person();

ar.sayAge();
```

```
public class Person {
    private int age;
    public Person() {
        age = in.nextInt();
    }
    public void sayAge() {
        System.out.println("My age is " + age);
}
}
```

More is needed:

- What if the attribute ‘age’ needs to be modified later?
- How can age be accessed but not just via a print()?

Viewing And Modifying Attributes

1) New terms: Accessor methods: ‘get()’ method
   - Used to determine the current value of an attribute
   - Example:

```
public int getAge()
{
   return(age);
}
```

2) New terms: Mutator methods: ‘set()’ method
   - Used to change an attribute (set it to a new value)
   - Example:

```
public void setAge(int anAge)
{
   age = anAge;
}
```

Class Person

```
public class Person
{
    private int age;
    private String name;

    public Person()
    {
        System.out.println("Person()");
        age = 0;
        name = "No-name";
    }
```

Class Person(2)

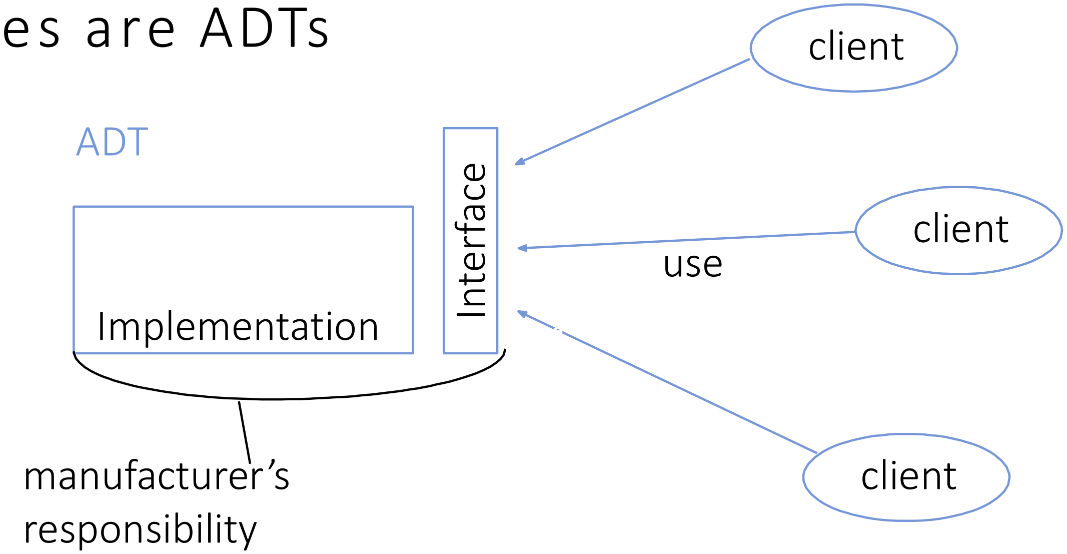
```
public Person(int anAge) {
   System.out.println("Person(int)");
   age = anAge;
   name = "No-name";
}
public Person(String aName) {
   System.out.println("Person(String)");
   age = 0;
   name = aName;
}
public Person(int anAge, String aName) {
   System.out.println("Person(int,String)");
   age = anAge;
   name = aName;
}
```

Class Person (3)

```
    public int getAge() {
        return(age);
    }

    public String getName() {
        return(name);
    }

    public void setAge(int anAge) {
        age = anAge;
    }
    public void setName(String aName) {
        name = aName;
    }
}
```

CLASS AS ADT

- Abstract Data Types (ADTs)
  - type implementation & operations
  - hidden implementation
- built-in and user-defined types are ADTs


client
ADT
Interface
client
use
Implementation
client
manufacturer's
responsibility

Message Passing

- It is a form of communication used in object-oriented programming as well as parallel programming.

- Objects communicate with one another by sending and receiving information to each other.

- A message for an object is a request for execution of a procedure and therefore will invoke a function in the receiving object that generates the desired results.

- Message passing involves specifying the name of the object, the name of the function, and the information to be sent.

Object

- It is a basic unit of Object-Oriented Programming and represents the real-life entities.
- An Object is an instance of a Class.
- When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.
- An object has an identity, state, and behavior.
- Each object contains data and code to manipulate the data.
- Objects can interact without having to know details of each other's data or code, it is sufficient to know the type of message accepted and type of response returned by the objects.

- For example "Dog" is a real-life Object, which has some characteristics like color, Breed, Bark, Sleep, and Eats.

New Term: Object State

- Similar to how two variables can contain different data.

- Attributes: Data that describes each instance or example of a class.

- Different objects have the same attributes but the values of those attributes can vary
    - Reminder: The class definition specifies the attributes and methods for *all objects*

- Example: two ‘monster’ objects each have a health attribute but the current value of their health can differ

- The current value of an object’s attribute’s determines it’s state.

C++ OBJECTS

- A class provides the blueprints for objects

- an object is created from a class

- Object is variable of class type

- Objects are declared the same way the variables are declared

```
Box      //    Declare Box1 of type Box
Box1;
         //    Declare Box2 of type Box
Box
Box2;
```

Both of the objects Box1 and Box2 will have their own copy of data members.

ACCESSING THE DATA MEMBERS

- The public data members of objects of a class can be accessed using the direct member access operator (.)

```
#include <iostream>
using namespace std;
class Box{
public:
    double length;// Length of a box
    double breadth; // Breadth of a box
    double height;   //Height of a box
    };
```

```
int main( ){
Box Box1;  //Declare Boz 1 of type Box
Box Box2;  // Declare Box2 of type Box
double volume = 0.0; //Store the volume of a box here
//box 1 specification
Box1.height = 5.0;
Box1.length = 6.0;
Box1.breadth = 7.0;//
 //box 2 specification
 Box2.height = 10.0;
 Box2.length = 12.0;
 Box2.breadth = 13.0;//
```

ACCESSING THE DATA MEMBERS

```
volume of // box 1
volume = Box1.height * Box1.length * Box1.breadth;
cout << "Volume of Box1 : " <<
volume <<endl;      //       volume
volume = Box2.height * Box2.length *Box2.breadth;
cout << "Volume of Box2 : " <<
volume <<endl;
return 0;
}
```

Classes/Objects

- Each class of object includes descriptive data.
    - Example (animals):
        - Species
        - Color
        - Length/height
        - Weight
        - Etc.
- Also each class of object has an associated set of actions
    - Example (animals):
        - Sleeping
        - Eating
        - Excreting
        - Etc.

Classes

- A class is a user-defined data type.

- It consists of data members and member functions, which can be accessed and used by creating an instance of that class.

- It represents the set of properties or methods that are common to all objects of one type. A class is like a blueprint for an object.

- *For Example:* Consider the Class of Cars.
- There may be many cars with different names and brands but all of them will share some common properties like all of them will have 4 wheels, Speed Limit, Mileage range, etc.

A Class Must Be First Defined

- A class is a new type of variable.

- The class definition specifies:

  - What descriptive data is needed?

    - Programming terminology: attributes = data (New definition)

  - What are the possible set of actions?

    - Programming terminology: methods = actions (new definition)

Why Must Classes Be Defined

- Some classes are already pre-defined (included) in a programming language with a list of attributes and methods e.g., String
- Why don’t more classes come ‘built’ into the language?
- The needs of the program will dictate what attributes and methods are needed.

Defining The Attributes Of A Class In Java

- Attributes can be variable or constant (preceded by the 'final' keyword), for now stick to the former.
- Format:

  <access modifier>[1] *<type of the attribute>* *<name of the attribute>*;

- Example:

```
public class Person
{
      private int age;
}
```

[1] 1) Although other options may be possible, *attributes are almost always set to private*.

MEMBERS OF CLASS

- A member function of a class is a function that has its definition or its prototype within the class
- Members are accessed using dot operator(.)

```
class Box
{
public:                          / Length of a box
 double length;                  / Breadth of a box   data member
                                 / Height of a box
 doublebreadth
 ;                               / Returns box volume
    doubleheight
    ;
  doublegetVolume(void);
  / member function
 };
```

Defining A Java Class

Format:

```
public class <name of class>
{
   attributes
     methods
}
```

Example (more explanations coming shortly):

```
   public class Person
 {
    private int age; // Attribute
    public Person() { // Method
        age = in.nextInt();
    }
    public void sayAge() {// Method
        System.out.println("My age is " + age);
    }
```

Class Person

```
public class Person
{
    public void sayHello()
    {
        System.out.println("I don't want to say hello.");
    }
}
```

MEMBERS OF CLASS(CONTINUED…)

- Member functions can be defined within the class definition or separately using scope resolution operator, ::

```
class Box{
  public:
   double length;                    / Length of a box
   double breadth;                   / Breadth of a box
   double height;                    / Height of a box
  double getVolume(void){
     return length * breadth
     * height;
     }
  };      // class end
```

MEMBERS OF CLASS(CONTINUED…)

- If you like you can define same function outside the class using

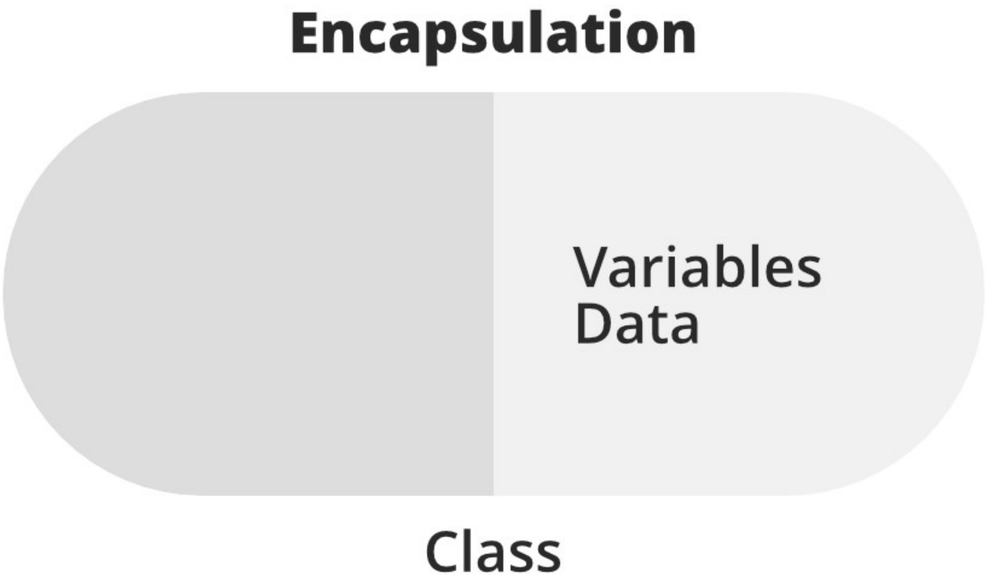
scope resolution operator, :: as follows:

```
double Box::getVolume(void)
{
return length * breadth * height;
}
```

Encapsulation

- Encapsulation is defined as the wrapping up of data under a single unit.
- It is the mechanism that binds together code and the data it manipulates.
- In Encapsulation, the variables or data of a class are hidden from any other class and can be accessed only through any member function of their class in which they are declared.
- As in encapsulation, the data in a class is hidden from other classes, so it is also known as data-hiding.

Encapsulation


Encapsulation
Variables
Data
Class

Encapsulation

- Consider a real-life example of encapsulation, in a company, there are different sections like the accounts section, finance section, sales section, etc.

- The finance section handles all the financial transactions and keeps records of all the data related to finance. Similarly, the sales section handles all the sales-related activities and keeps records of all the sales.

- Now there may arise a situation when for some reason an official from the finance section needs all the data about sales in a particular month.

- In this case, he is not allowed to directly access the data of the sales section.

- He will first have to contact some other officer in the sales section and then request him to give the particular data.

DATA ENCAPSULATION

- The wrapping up of data and function into a single unit (called class) is known as encapsulation.

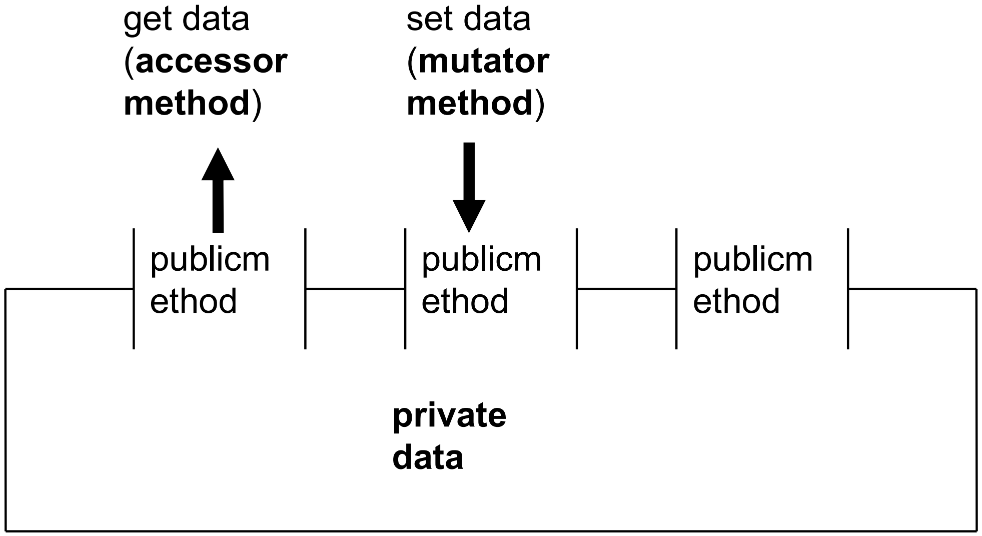
- Data and encapsulation is the most striking feature of a class.

- OOP encapsulates data (attributes) and functions (behavior) into packages called objects

- E.g. class encapsulates data members and member functions

New Term: Encapsulation/Information Hiding

- Protects the inner-workings (data) of a class.
- Only allow access to the core of an object in a controlled fashion (use the *public* parts to access the *private* sections).
  - Typically it means public methods accessing private attributes via accessor and mutator methods.
  - Controlled access to attributes:
    - Can prevent invalid states
    - Reduce runtime errors


get data (**accessor method**)
set data (**mutator method**)
publicm ethod
publicm ethod
publicm ethod
private data

Encapsulation

Microsoft .net

Encapsulate

Encapsulation

- Encapsulation hides the implementation details
- Class announces some operations (methods) available for its clients – its public interface
- All data members (fields) of a class should be hidden
    - Accessed via properties (read-only and read-write)
- No interface members should be hidden

Encapsulation – Example

- Data fields are private
- Constructors and accessors are defined (getters and setters)

Person

-name : string
-age : TimeSpan

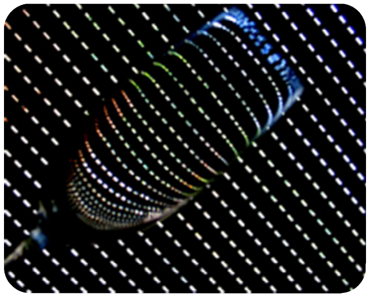
+Person(string name, int age)
+Name : string { get; set; }
+Age : TimeSpan { get; set; }

Encapsulation in .NET

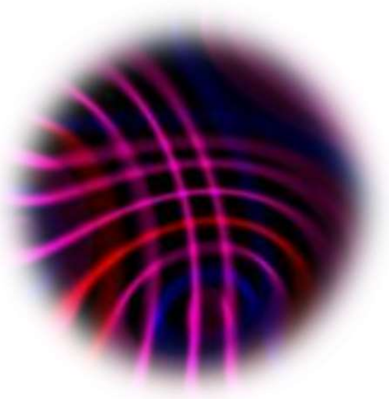
- Fields are always declared private
    - Accessed through properties in read-only or read-write mode
- Constructors are almost always declared public
- Interface methods are always public
    - Not explicitly declared with public
- Non-interface methods are declared private / protected

Encapsulation – Benefits

- Ensures that structural changes remain local:
  - Changing the class internals does not affect any code outside of the class
  - Changing methods' implementation
  - does not reflect the clients using them
- Encapsulation allows adding some logic when accessing client's data
    - E.g. validation on modifying a property value
- Hiding implementation details reduces complexity → easier maintenance

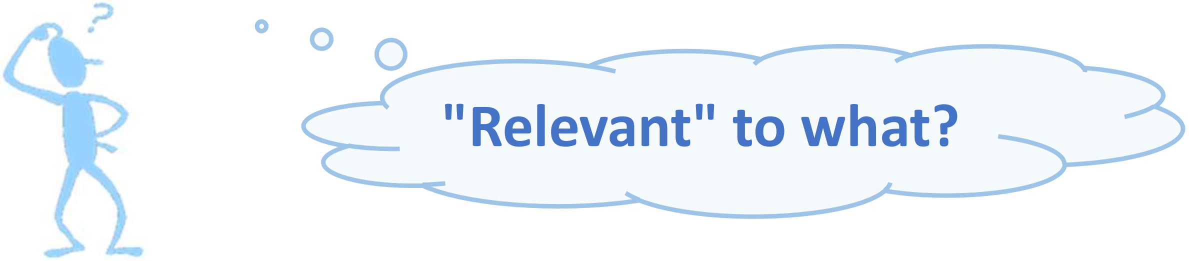


Abstraction



Abstraction

- Abstraction means ignoring irrelevant features, properties, or functions and emphasizing the relevant ones ...


"Relevant" to what?

- ... relevant to the given project (with an eye to future reuse in similar projects)

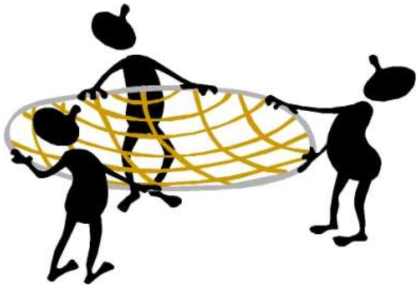
- Abstraction = managing complexity

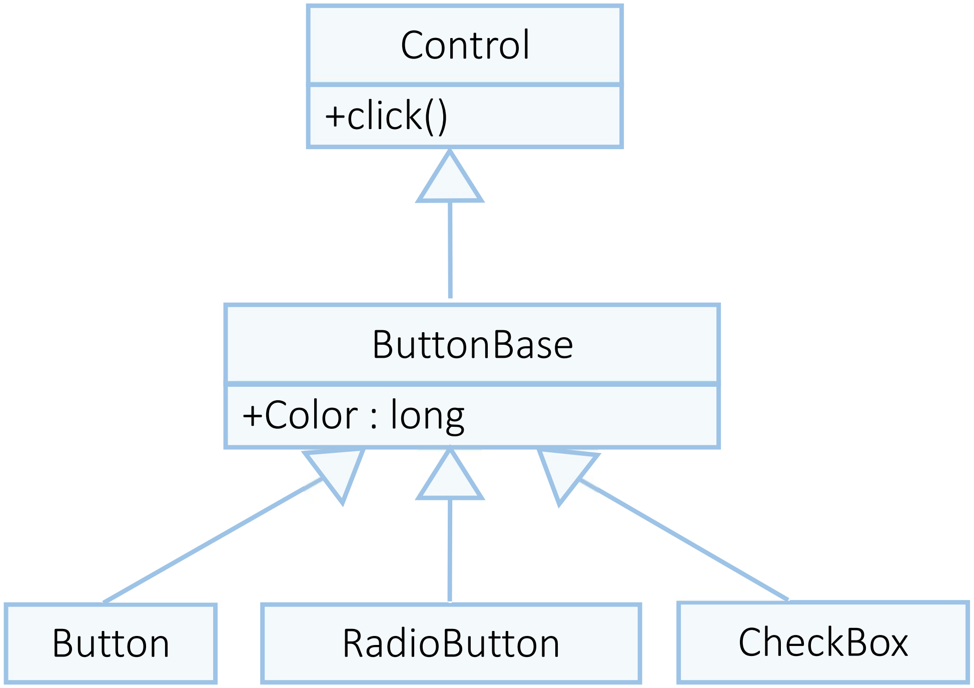
Abstraction (2)

- Abstraction is something we do every day
  - Looking at an object, we see those things about it that have meaning to us
  - We abstract the properties of the object, and keep only what we need
  - E.g. students get "name" but not "color of eyes“
- Allows us to represent a complex reality in terms of a simplified model
- Abstraction highlights the properties of an entity that we need and hides the others

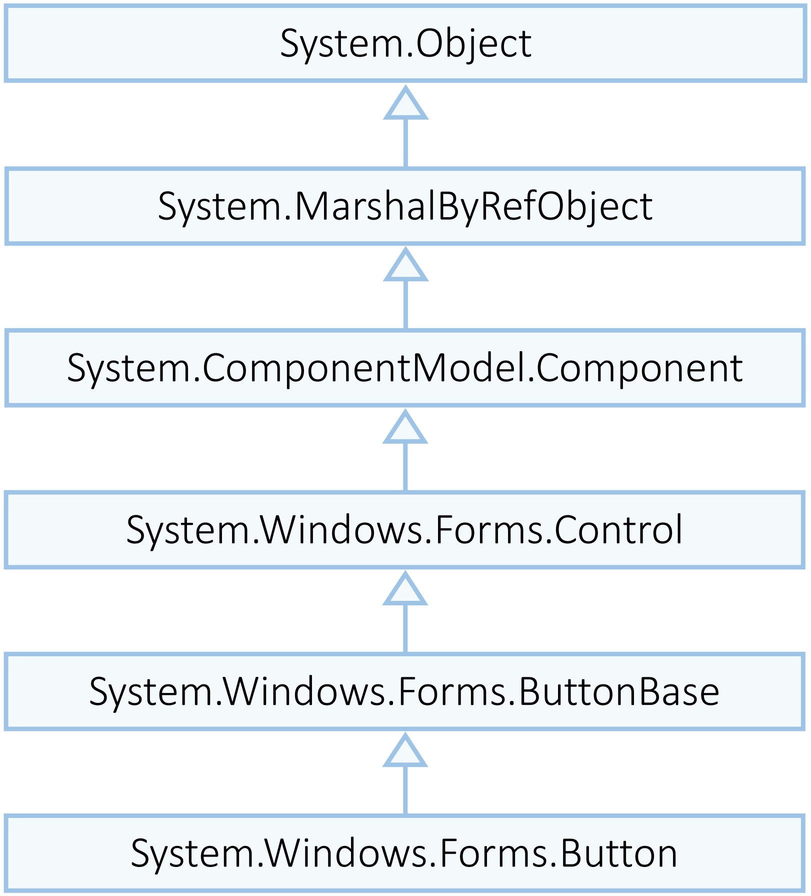
Abstraction in .NET

- In .NET abstraction is achieved in several ways:
  - Abstract classes
  - Interfaces
  - Inheritance




Control
+click()
ButtonBase
+Color : long
Button
RadioButton
CheckBox

Abstraction in .NET – Example



System.Object

System.MarshalByRefObject

System.ComponentModel.Component

System.Windows.Forms.Control

System.Windows.Forms.ButtonBase

System.Windows.Forms.Button

Interfaces in C#

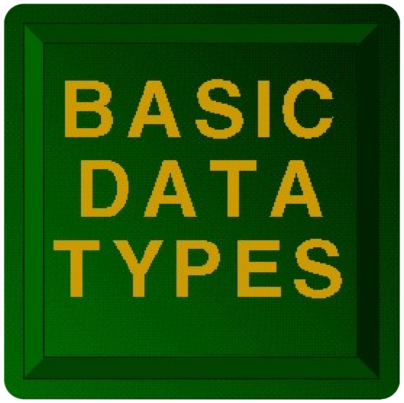
- An interface is a set of operations (methods) that given object can perform
  - Also called "contract" for supplying a set of operations
  - Defines abstract behavior
- Interfaces provide abstractions
  - You shouldn't have to know anything about what is in the implementation in order to use it

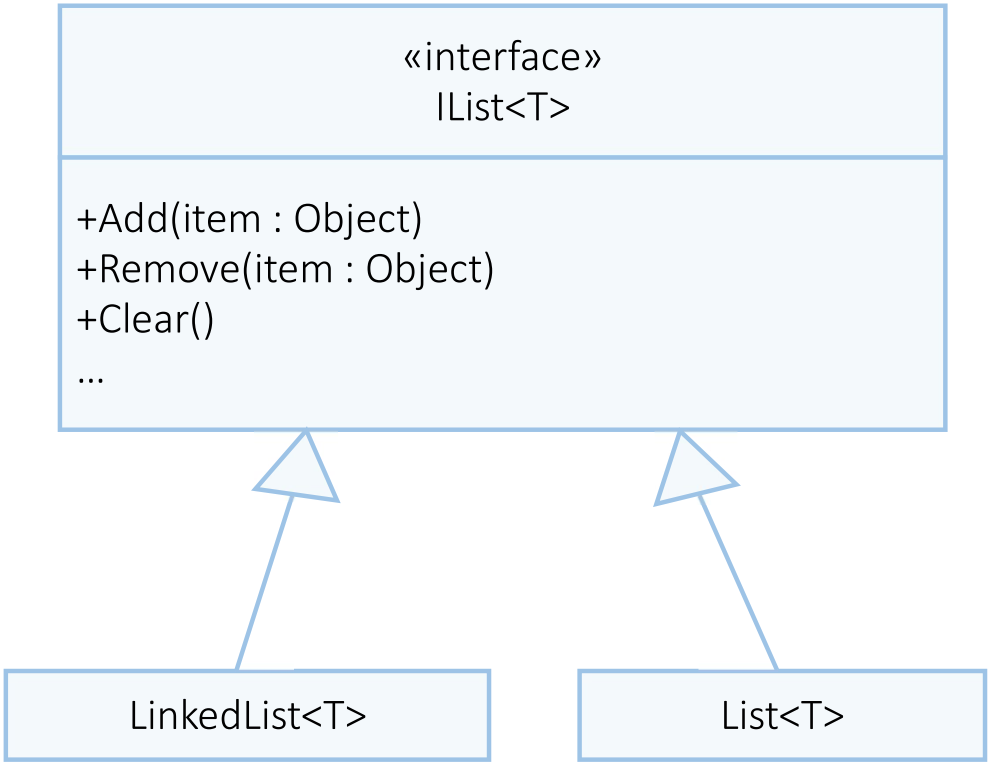
Abstract Classes in C#

- Abstract classes are special classes defined with the keyword abstract
  - Mix between class and interface
  - Partially implemented or fully unimplemented
  - Not implemented methods are declared abstract and are left empty
  - Cannot be instantiated
- Child classes should implement abstract methods or declare them as abstract

Abstract Data Types

- Abstract Data Types (ADT) are data types defined by a set of operations (interface)
- Example:





«interface»
IList<T>

+Add(item : Object)
+Remove(item : Object)
+Clear()
…

LinkedList<T>

List<T>

Data Abstraction

- Data abstraction is one of the most essential and important features of object-oriented programming.

- Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.

- Consider a real-life example of a man driving a car.

- The man only knows that pressing the accelerators will increase the speed of the car or applying brakes will stop the car, but he does not know about how on pressing the accelerator the speed is increasing, he does not know about the inner mechanism of the car or the implementation of the accelerator, brakes, etc., in the car.

- This is what abstraction is.

DATA ABSTRACTION

- Abstraction refers to the act of representing essential features without including the background details or explanation

- Classes use the concept of abstraction and are defined as a list of abstract attributes such as size, wait, and cost, and function operate on these attributes.

- E.g TV
  - a television separates its internal implementation from its external interface and we can play with its interfaces like the power button, channel changer, and volume control without having any knowledge of its internals .

INFORMATION HIDING

- Data hiding is one of the important features of Object Oriented Programming which allows preventing the functions of a program to access directly the internal representation of a class type
- The access restriction to the class members is specified by the labeled **public, private,** and **protected** sections within the class body
- private, and protected are called access specifiers/modifiers.
- A class can have multiple public, protected, or private labeled sections
- The default access for members and classes is private

INFORMATION HIDING(CONTINUED)

```
class Base {
  public: /public members go here
  protected:/    protected members go here
  private:/      private members go here
  };
```

INFORMATION HIDING(CONTINUED)

- A public member is accessible from anywhere outside the class but within a program

- A private member variable or function cannot be accessed, or even viewed from outside the class.

- Only the class and friend functions can access private members.

- By default all the members of a class would be private

- A protected member variable or function is very similar to a private member but it provided one additional benefit that they can be accessed in child classes which are called derived classes

Inheritance

Inheritance is an important pillar of OOP(Object-Oriented Programming).

The capability of a class to derive properties and characteristics from another class is called Inheritance.

When we write a class, we inherit properties from other classes.

So when we create a class, we do not need to write all the properties and functions again and again, as these can be inherited from another class that possesses it.

Inheritance allows the user to reuse the code whenever possible and reduce its redundancy.

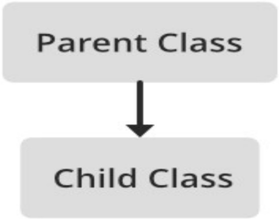
INHERITANCE

- Inheritance is the process by which objects of one class acquired the properties of objects of another classes.

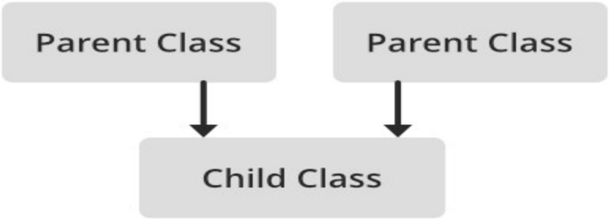
- In OOP, the concept of inheritance provides the idea of reusability

- add additional features to an existing class without modifying it

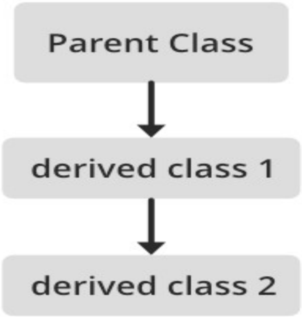
Example


Parent Class
Child Class

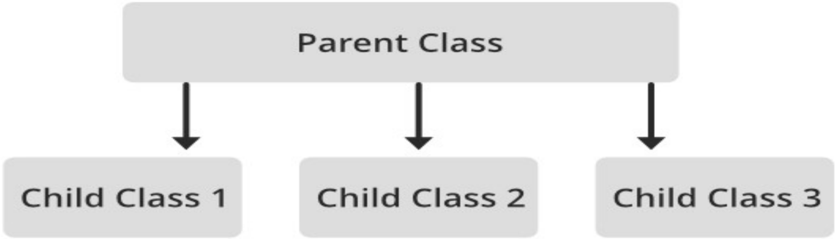
Single inheritance


Parent Class
Parent Class
Child Class

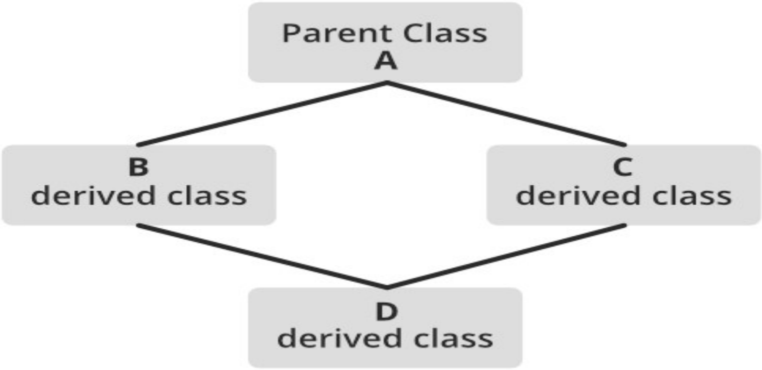
Multiple inheritance


Parent Class
derived class 1
derived class 2

Multilevel inheritance


Parent Class
Child Class 1
Child Class 2
Child Class 3

Hierarchical inheritance


Parent Class A
B derived class
C derived class
D derived class

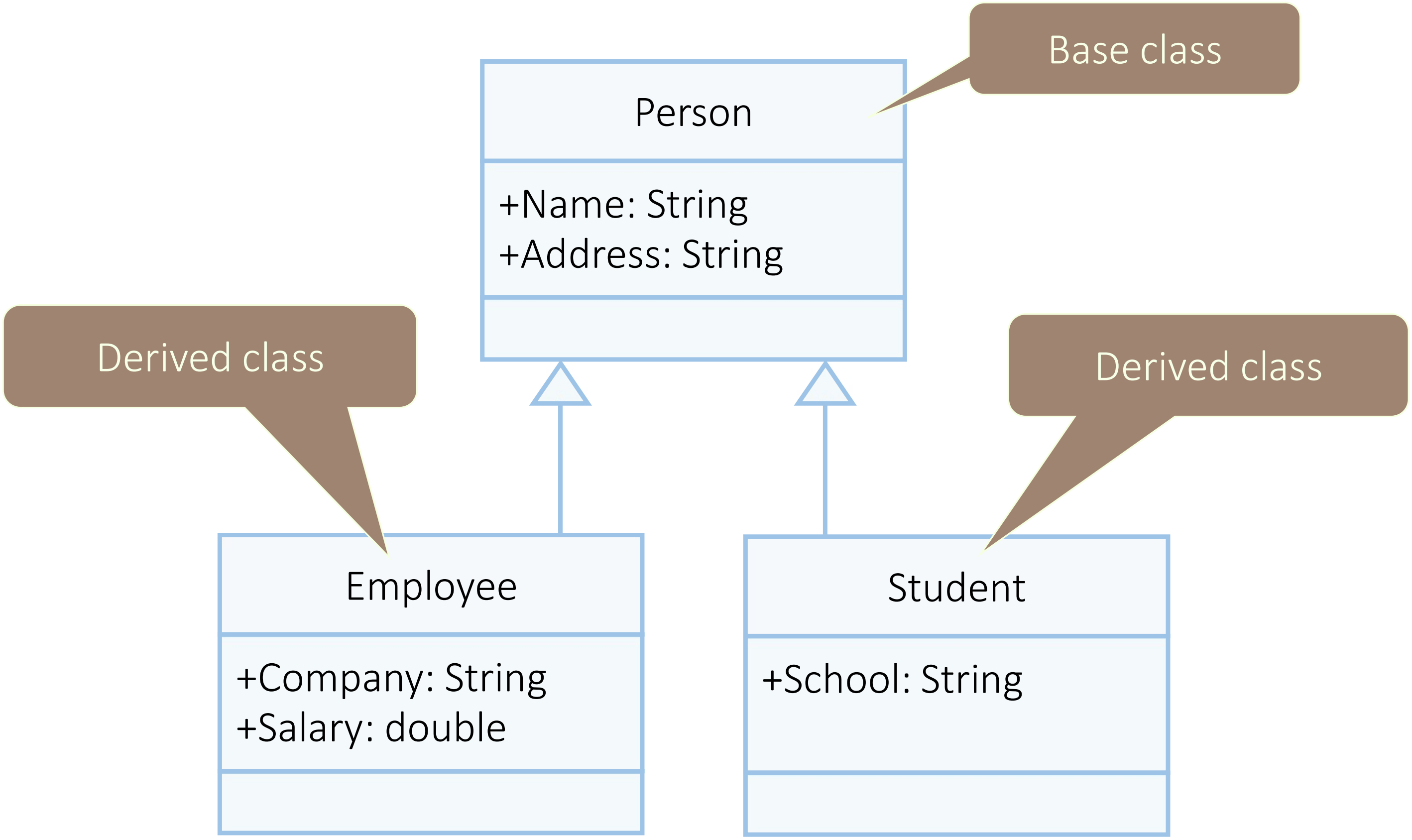
Hybrid Class

Inheritance – Benefits

- Inheritance has a lot of benefits
  - Extensibility
  - Reusability
  - Provides abstraction
  - Eliminates redundant code
- Use inheritance for buidling is-a relationships
  - E.g. dog is-a animal (dogs are kind of animals)
- Don't use it to build has-a relationship
  - E.g. dog has-a name (dog is not kind of name)

Big
BENEFITS
Package

Inheritance – Example



Base class

Person

+Name: String
+Address: String

Derived class

Employee

+Company: String
+Salary: double

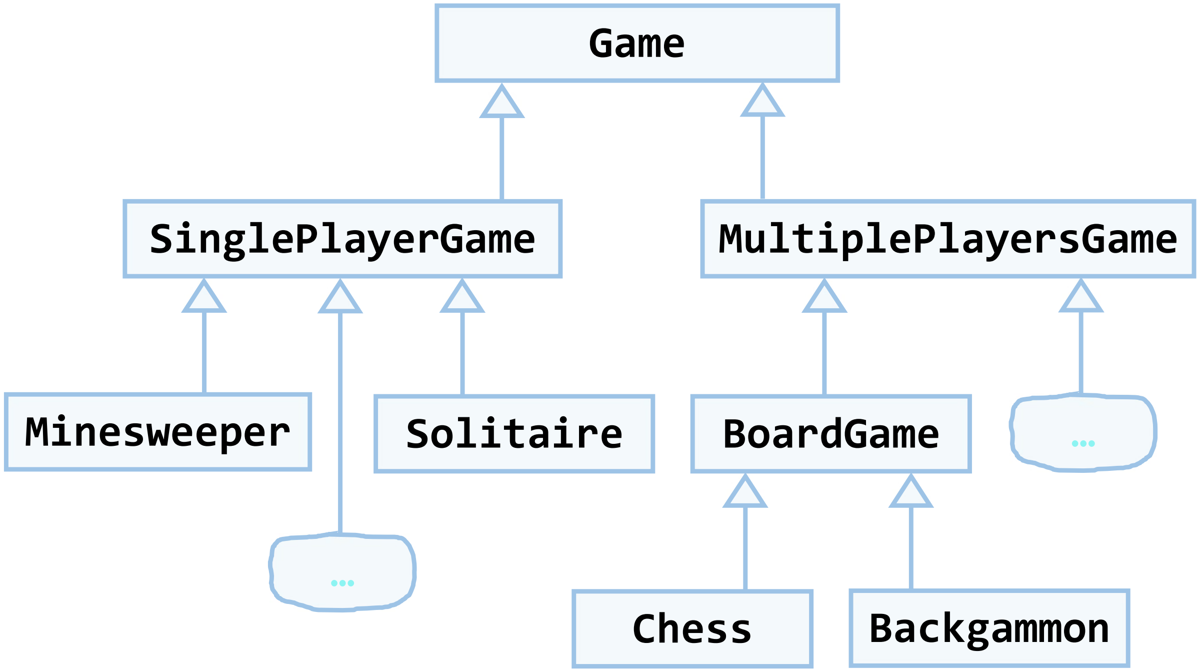
Derived class

Student

+School: String

Class Hierarchies

- Inheritance leads to a hierarchy of classes and/or interfaces in an application:


Game
SinglePlayerGame
MultiplePlayersGame
Minesweeper
Solitaire
BoardGame
...
...
Chess
Backgammon

Inheritance in .NET

- A class can inherit only one base class
  - E.g. IOException derives from SystemException and it derives from Exception
- A class can implement several interfaces
  - This is .NET’s form of multiple inheritance
  - E.g. List<T> implements IList<T>, ICollection<T>, IEnumerable<T>
- An interface can implement several interfaces
  - E.g. IList<T> implements ICollection<T> and IEnumerable<T>

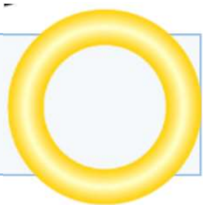
How to Define Inheritance?

- We must specify the name of the base class after the name of the derived

```
public class Shape
{...}
public class Circle : Shape
{...}
```

- In the constructor of the derived class we use the keyword base to invoke the constructor of the base class

```
public Circle (int x, int y) : base(x)
{...}
```



Simple Inheritance Example

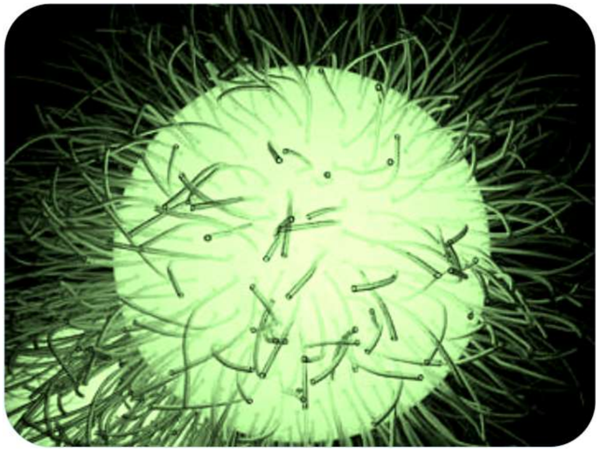
```
public class Mammal
{
  public int Age { get; set; }

  public Mammal(int age)
  {
    this.Age = age;
  }

  public void Sleep()
  {
    Console.WriteLine("Shhh! I'm sleeping!");
  }
}
```

Simple Inheritance Example (2)

```
public class Dog : Mammal
{
  public string Breed { get; set; }

  public Dog(int age, string breed)
    : base(age)
  {
    this.Breed = breed;
  }

  public void WagTail()
  {
    Console.WriteLine("Tail wagging...");
  }
}
```

Simple Inheritance

INHERITANCE



Accessibility Levels

- Access modifiers in C#
  - public – access is not restricted
  - private – access is restricted to the containing type
  - protected – access is limited to the containing type and types derived from it
  - internal – access is limited to the current assembly
  - protected internal – access is limited to the current assembly or types derived from the containing class

Inheritance and Accessibility

```
class Creature
{
  protected string Name { get; private set; }

  private void Talk()
  {
    Console.WriteLine("I am creature ...");
  }

  protected void Walk()
  {
    Console.WriteLine("Walking ...");
  }
}

class Mammal : Creature
{
  // base.Talk() can be invoked here
  // this.Name can be read but cannot be modified here
}
```

Inheritance and Accessibility (2)

```
class Dog : Mammal
{
  public string Breed { get; private set; }
  // base.Talk() cannot be invoked here (it is private)
}
class InheritanceAndAccessibility
{
  static void Main()
  {
    Dog joe = new Dog(6, "Labrador");
    Console.WriteLine(joe.Breed);
    // joe.Walk() is protected and can not be invoked
    // joe.Talk() is private and can not be invoked
    // joe.Name = "Rex"; // Name cannot be accessed here
    // joe.Breed = "Shih Tzu"; // Can't modify Breed
  }
}
```

Inheritance and Accessibility

Inheritance: Important Aspects

- Structures cannot be inherited

- In C# there is no multiple inheritance
  - Only multiple interfaces can be implemented

- Instance and static constructors are not inherited

- Inheritance is transitive relation
  - If C is derived from B, and B is derived from A, then C inherits A as well

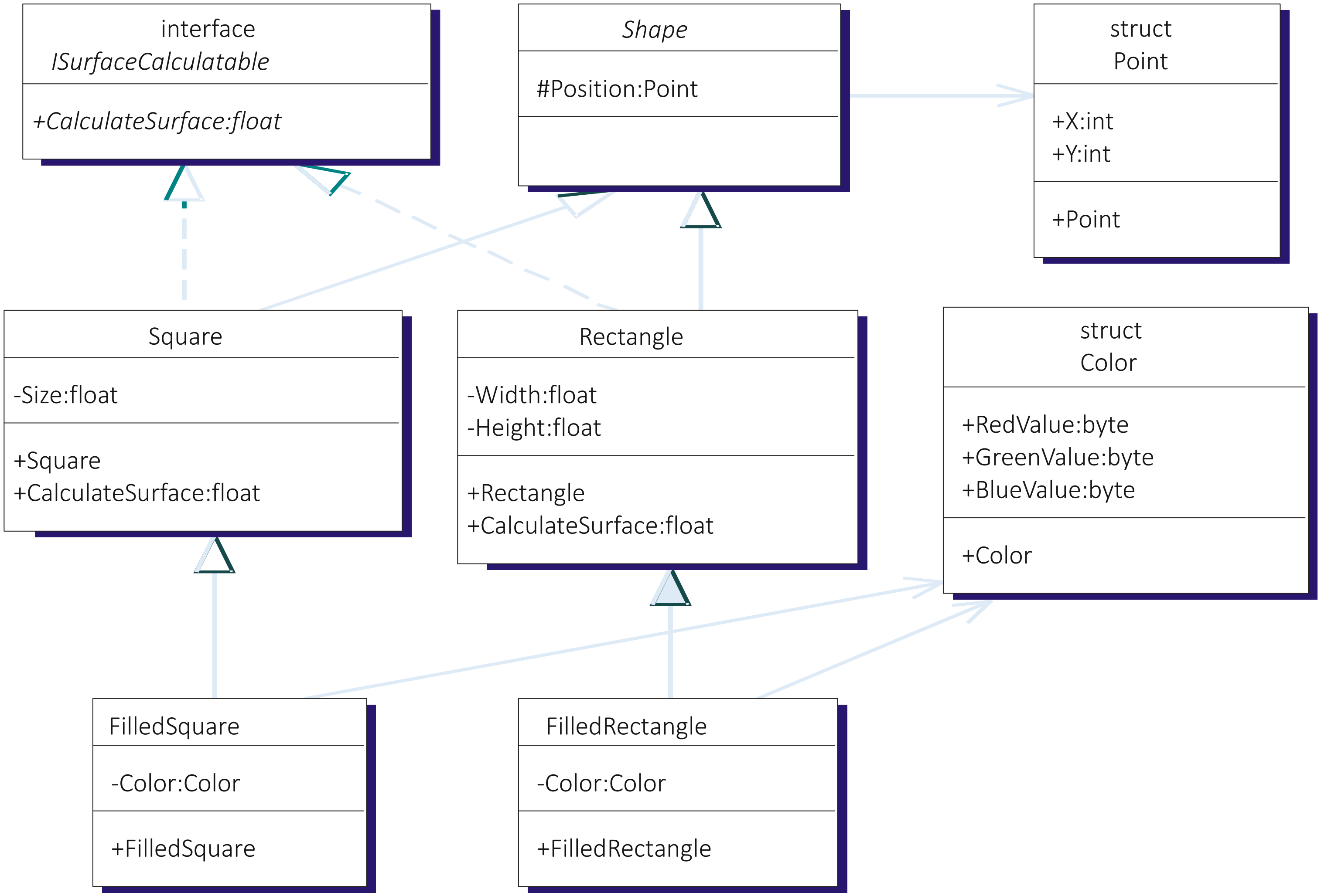
Inheritance: Important Features

- A derived class extends its base class
  - It can add new members but cannot remove derived ones
- Declaring new members with the same name or signature hides the inherited ones
- A class can declare virtual methods and properties
  - Derived classes can override the implementation of these members
  - E.g. Object.Equals() is virtual method

Inheritance Hierarchies

- Using inheritance we can create inheritance hierarchies
    - Easily represented by UML class diagrams
- UML class diagrams
    - Classes are represented by rectangles containing their methods and data
    - Relations between classes are shown as arrows
        - Closed triangle arrow means inheritance
        - Other arrows mean some kind of associations

UML Class Diagram – Example



interface
ISurfaceCalculatable

+CalculateSurface:float

Shape

#Position:Point

struct
Point

+X:int
+Y:int

+Point

Square

-Size:float

+Square
+CalculateSurface:float

Rectangle

-Width:float
-Height:float

+Rectangle
+CalculateSurface:float

struct
Color

+RedValue:byte
+GreenValue:byte
+BlueValue:byte

+Color

FilledSquare

-Color:Color

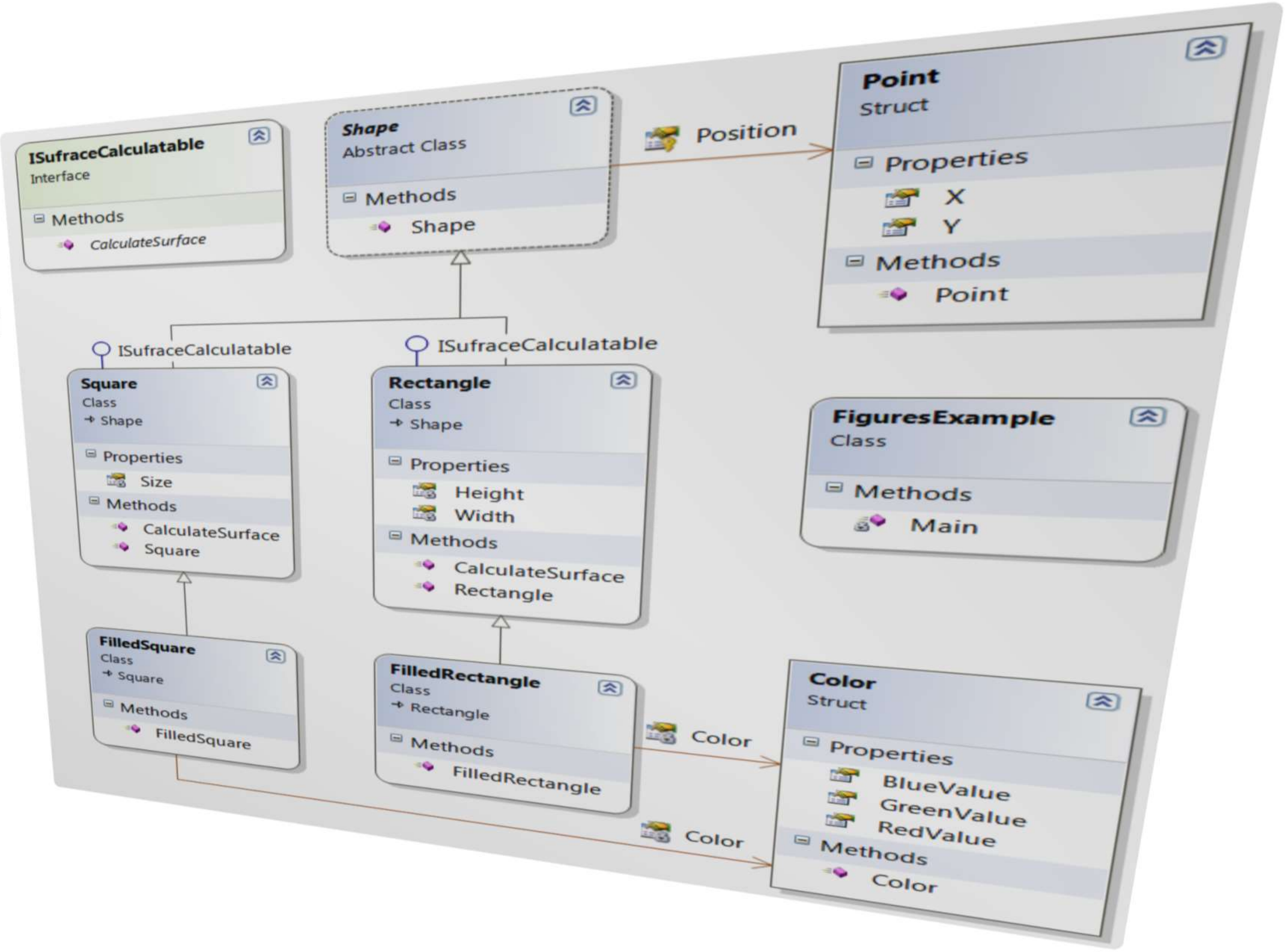
+FilledSquare

FilledRectangle

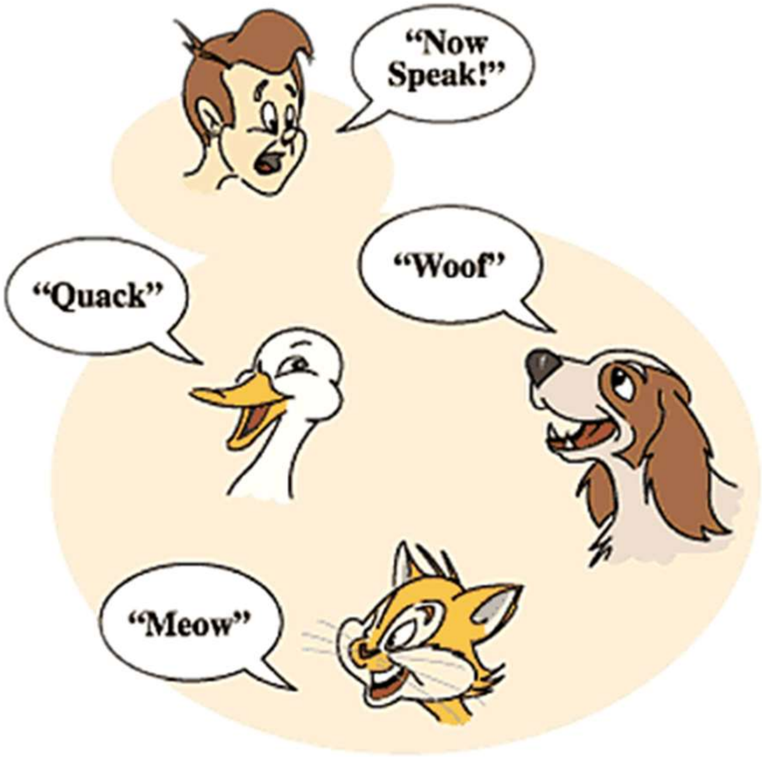
-Color:Color

+FilledRectangle

Class Diagrams in Visual Studio


ISufraceCalculatable
Interface
Methods
CalculateSurface
Shape
Abstract Class
Methods
Shape
Position
Point
Struct
Properties
X
Y
Methods
Point
ISufraceCalculatable
ISufraceCalculatable
Square
Class
Shape
Properties
Size
Methods
CalculateSurface
Square
Rectangle
Class
Shape
Properties
Height
Width
Methods
CalculateSurface
Rectangle
FiguresExample
Class
Methods
Main
FilledSquare
Class
Square
Methods
FilledSquare
FilledRectangle
Class
Rectangle
Methods
FilledRectangle
Color
Color
Color
Struct
Properties
BlueValue
GreenValue
RedValue
Methods
Color

Polymorphism


"Now Speak!"
"Woof"
"Quack"
"Meow"

Polymorphism

- Polymorphism = ability to take more than one form (objects have more than one type)
  - A class can be used through its parent interface
  - A child class may override some of the behaviors of the parent class
- Polymorphism allows abstract operations to be defined and used
  - Abstract operations are defined in the base class' interface and implemented in the child classes
    - Declared as abstract or virtual

Polymorphism

- Why handle an object of given type as object of its base type?
  - To invoke abstract operations
  - To mix different related types in the same collection
    - E.g. List<object> can hold anything
  - To pass more specific object to a method that expects a parameter of a more generic type
  - To declare a more generic field which will be initialized and "specialized" later

Virtual Methods

- Virtual method is method that can be used in the same way on instances of base and derived classes but its implementation is different

- A method is said to be a virtual when it is declared as virtual

```
public virtual void CalculateSurface()
```

- Methods that are declared as virtual in a base class can be overridden using the keyword override in the derived class

The override Modifier

- Using override we can modify a method or property
- An override method provides a new implementation of a member inherited from a base class
- You cannot override a non-virtual or static method
- The overridden base method must be virtual, abstract, or override

Polymorphism – How it Works?

- Polymorphism ensures that the appropriate method of the subclass is called through its base class's interface

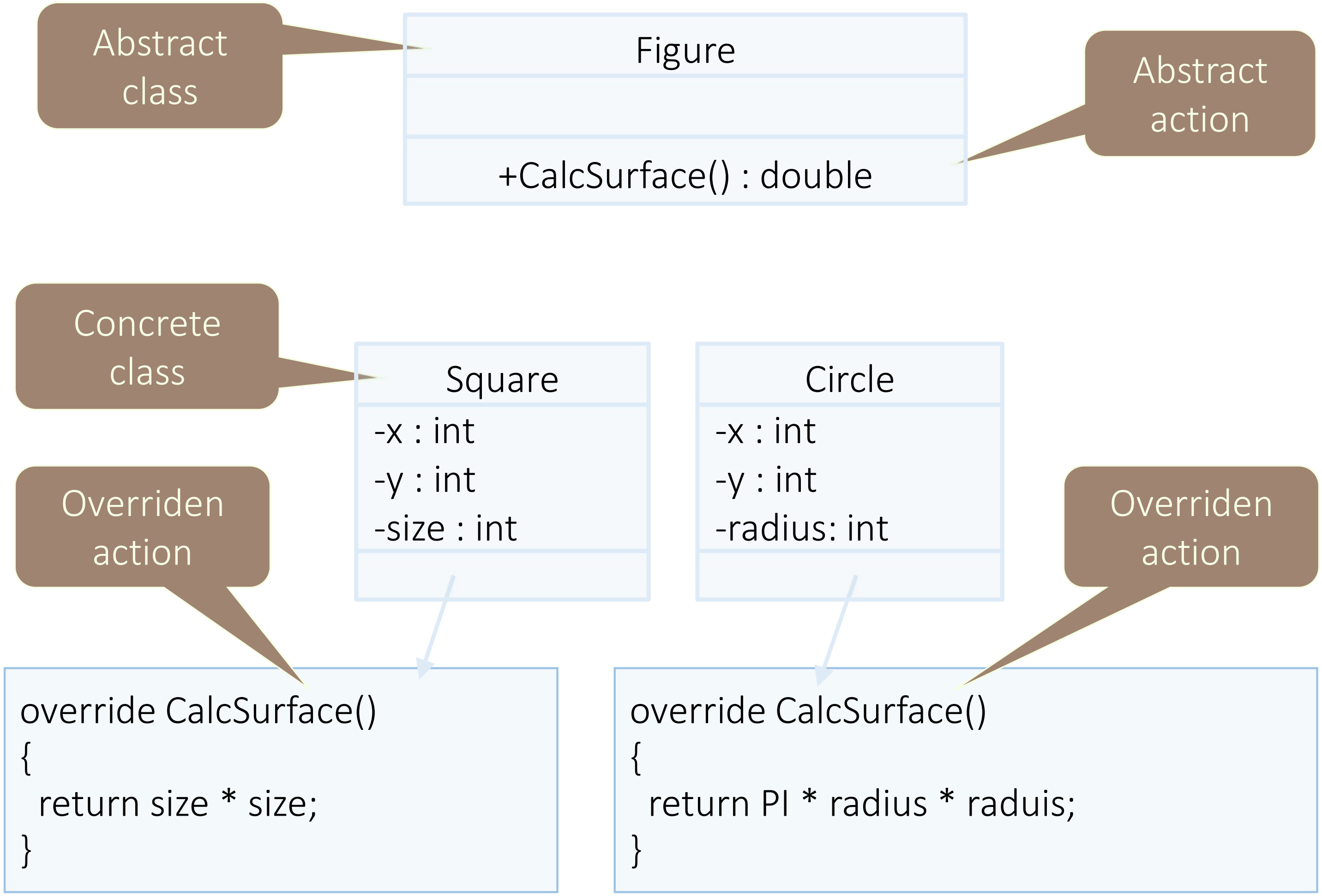
- Polymorphism is implemented using a technique called late method binding

  - Exact method to be called is determined at runtime, just before performing the call

  - Applied for all abstract / virtual methods

- Note: Late binding is slower than normal (early) binding

Polymorphism – Example



Abstract class

Figure

+CalcSurface() : double

Abstract action

Concrete class

Square

-x : int
-y : int
-size : int

Circle

-x : int
-y : int
-radius: int

Overriden action

```
override CalcSurface()
{
  return size * size;
}
```

Overriden action

```
override CalcSurface()
{
  return PI * radius * raduis;
}
```

Polymorphism – Example (2)

```
abstract class Figure
{
  public abstract double CalcSurface();
}

abstract class Square
{
  public override double CalcSurface() { return … }
}

Figure f1 = new Square(...);
Figure f2 = new Circle(...);

// This will call Square.CalcSurface()
int surface = f1.CalcSurface();

// This will call Square.CalcSurface()
int surface = f2.CalcSurface();
```

Polymorphism

POLYMORPHISM- Contd..

- Polymorphism is another important OOP concept.
- Polymorphism, a Greek term, means the ability to take more than on form
- An operation may exhibit different behavior is different instances.
- The process of making an operator to exhibit different behaviors in different instances is known as operator overloading.
- Using a single function name to perform different type of task is known as function overloading.
- E.g. abs(parameter type) instead of abs(), abs()

Polymorphism

The word polymorphism means having many forms.

In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.

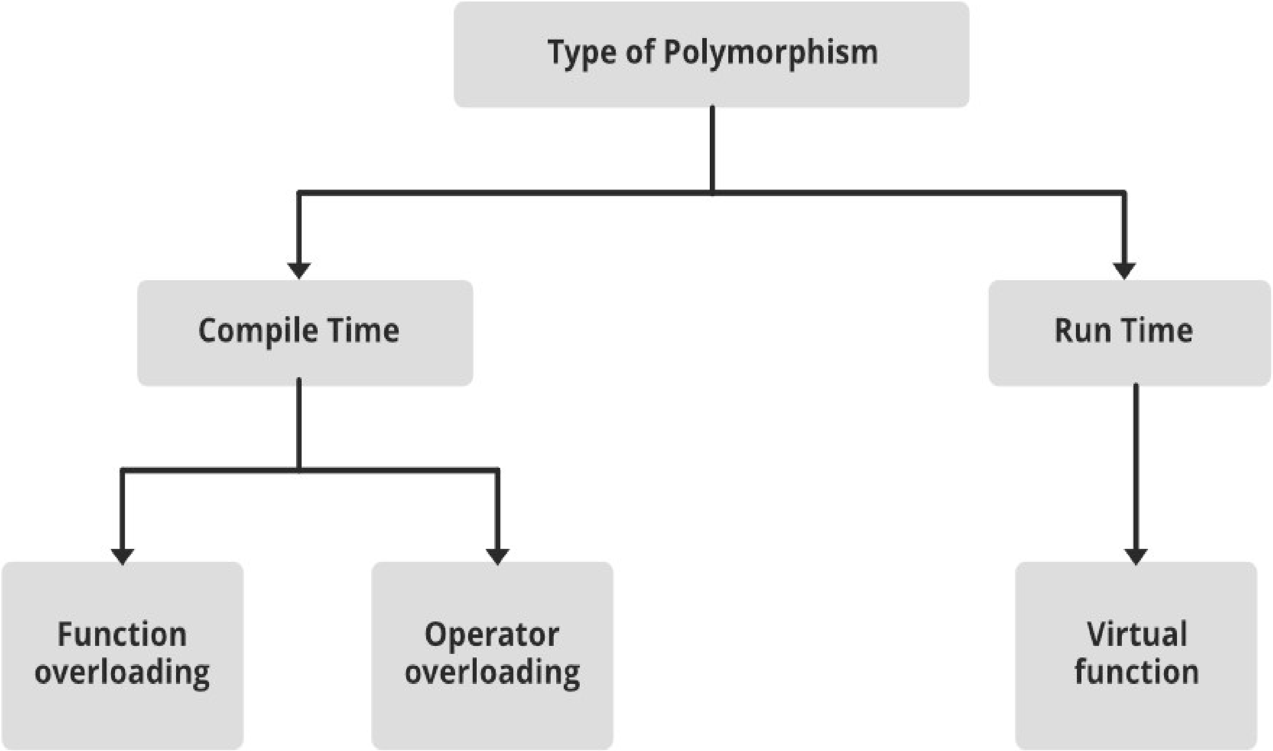
For example, A person at the same time can have different characteristics.

Like a man at the same time is a father, a husband, an employee.

So the same person posses different behavior in different situations.

This is called polymorphism.

Types of Polymorphism



Type of Polymorphism

Compile Time

Run Time

Function overloading

Operator overloading

Virtual function

Dynamic Binding

In dynamic binding, the code to be executed in response to the function call is decided at runtime.

Dynamic binding means that the code associated with a given procedure call is not known until the time of the call at run time.

Dynamic Method Binding One of the main advantages of inheritance is that some derived class D has all the members of its base class B.

Once D is not hiding any of the public members of B, then an object of D can represent B in any context where a B could be used.

This feature is known as subtype polymorphism.

MESSAGE PASSING

- An object-oriented program consists of a set of objects that communicate with each other.
- The process of programming in an object-oriented language, involves the following basic steps:
  - Creating classes that define object and their behaviour,
  - Creating objects from class definitions, and
  - Establishing communication among objects.
- A Message for an object is a request for execution of a procedure, and therefore will invoke a function (procedure) in the receiving object that generates the desired results.
- Message passing involves specifying the name of object, the name of the function (message) and the information to be sent

New Terminology: Method Signature

- Method signatures consist of: the type, number and order of the parameters.

- The signature will determine the overloaded method called:

```
Person p1 = new Person();

Person p2 = new Person(25);
```

Overloading and Good Design

- Overloading: methods that implement similar but not identical tasks.

- Examples include class constructors but this is not the only type of overloaded methods:

  System.out.println(int)
  System.out.println(double)
  etc.

  For more details on class System see:

  - http://java.sun.com/j2se/1.5.0/docs/api/java/io/PrintStream.html

- Benefit: just call the method with required parameters.

Method Overloading: Things To Avoid

- Distinguishing methods solely by the order of the parameters.

```
m(int,char);
  Vs.
m( char, int );
```

- Overloading methods but having an identical implementation.

- Why are these things bad?

Method Signatures And Program Design

- Unless there is a compelling reason do not change the signature of your methods!

Before:

```
class Foo
{
    void
fun()
    {

    }
}
```

```
public static void main ()
{
    Foo f = new Foo();
    f.fun()
}
```

This change is not good! ☹

After:

```
class Foo
{
    void
fun(int num)
    {

    }
}
```

Back To The ‘Private’ Keyword

- It syntactically means this part of the class cannot be accessed outside of the class definition.
  - You should always do this for variable attributes, *very rarely do this* for methods (more later).
- Example

```
public class Person {
    private int age;
    public Person() {
        age = 12;
    }
}
public class Driver {
    public static void main(String [] args) {
        Person aPerson = new Person();
        aPerson.age = 12;

    }
}
```

Class Inventory

```
public class Inventory
{
    public int stockLevel;

    public Inventory()
    {
        stockLevel = 0;
    }
}
```

Class Inventory

```
public class Inventory
{
    public final int CRITICAL = 10;
    public final int MIN = 0;
    public final int MAX = 100;
    private int stockLevel = 0;

    public boolean inventoryTooLow()
    {
          if (stockLevel < CRITICAL)
               return(true);
          else
               return(false);
    }
```

Class Inventory (2)

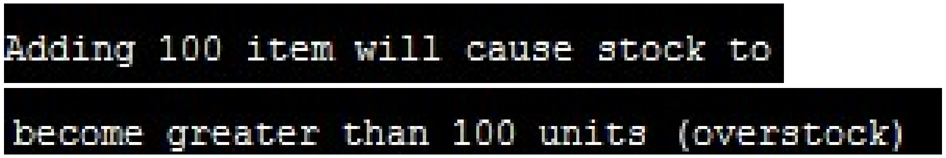
```
public void add(int amount)
{
   int temp;
   temp = stockLevel + amount;
   if (temp > MAX)
   {
      System.out.println();
      System.out.print("Adding " + amount + " item will cause stock ");
      System.out.println("to become greater than " + MAX + " units (overstock)");
   }
   else
   {
      stockLevel = temp;
   }
}
```

Class Inventory (3)

```
public void remove(int amount)
{
    int temp;
      temp = stockLevel - amount;
    if (temp < MIN)
    {
            System.out.print("Removing " + amount + " item will cause stock ");
               System.out.println("to become less than " + MIN + " units (understock)");
    }
      else
    {
        stockLevel = temp;
    }
}
public String showStockLevel ()
{ return("Inventory: " + stockLevel); }
}
```

Add(): Try Adding 100 items to 20 items

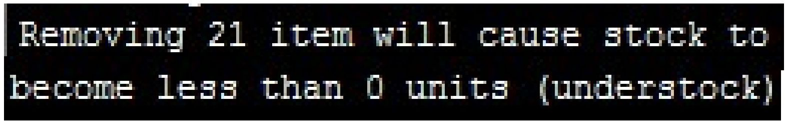
```
public void add(int amount)
{
    int temp;
    temp = stockLevel + amount;
    if (temp > MAX)
    {
        System.out.println();
        System.out.print("Adding " + amount + " item will cause stock ");
        System.out.println("to become greater than " + MAX + " units (overstock)");
    }
    else
    {
        stockLevel = temp;
    }
} // End of method add
```



```
Adding 100 item will cause stock to
become greater than 100 units (overstock)
```

Remove(): Try To Remove 21 Items From 20 Items

```
    public void remove(int amount)
     {
        int temp;
          temp = stockLevel - amount;
        if (temp < MIN)
        {
            System.out.println("Removing " + amount + " item will cause stock ");
            System.out.println("to become less than " + MIN + " units (understock)");
        }
          else
        {
            stockLevel = temp;
        }
     }
     public String showStockLevel ()
     { return("Inventory: " + stockLevel); }
 }
```



```
Removing 21 item will cause stock to
become less than 0 units (understock)
```

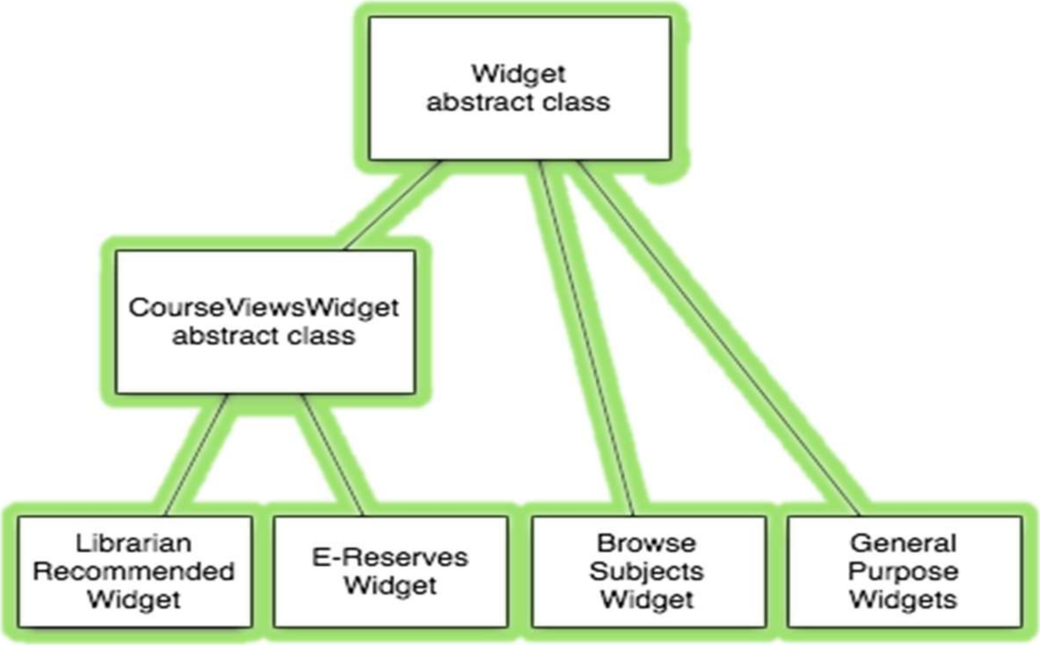
New Terms And Definitions

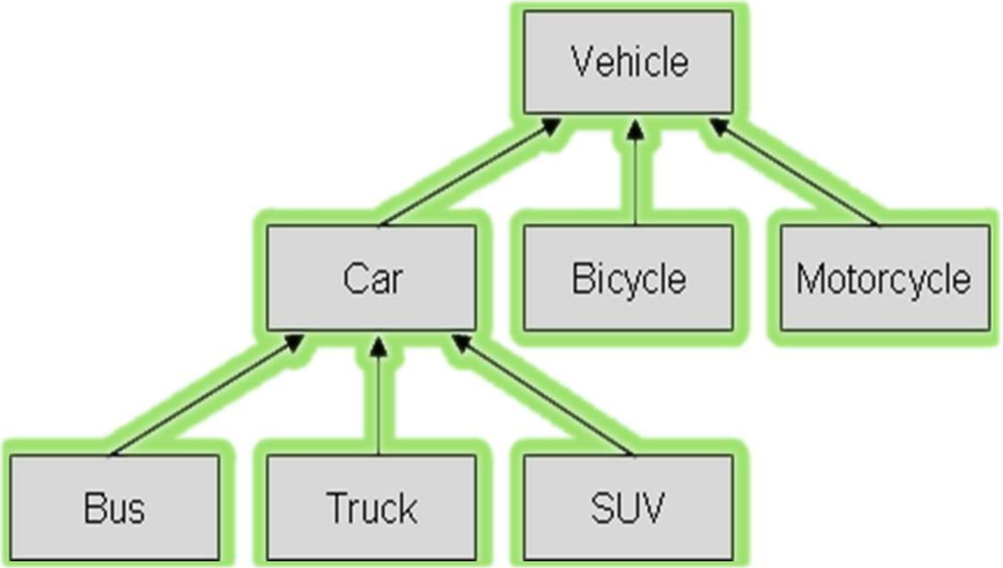
- Object-Oriented programming
- Class
- Object
- Class attributes
- Class methods
- Object state
- Instantiation
- Constructor (and the Default constructor)
- Method
- Function

New Terms And Definitions (2)

- Accessor method (“get”)
- Mutator method (“set”)
- Method overloading
- Method signature
- Encapsulation/information hiding
- Multiplicity/cardinality

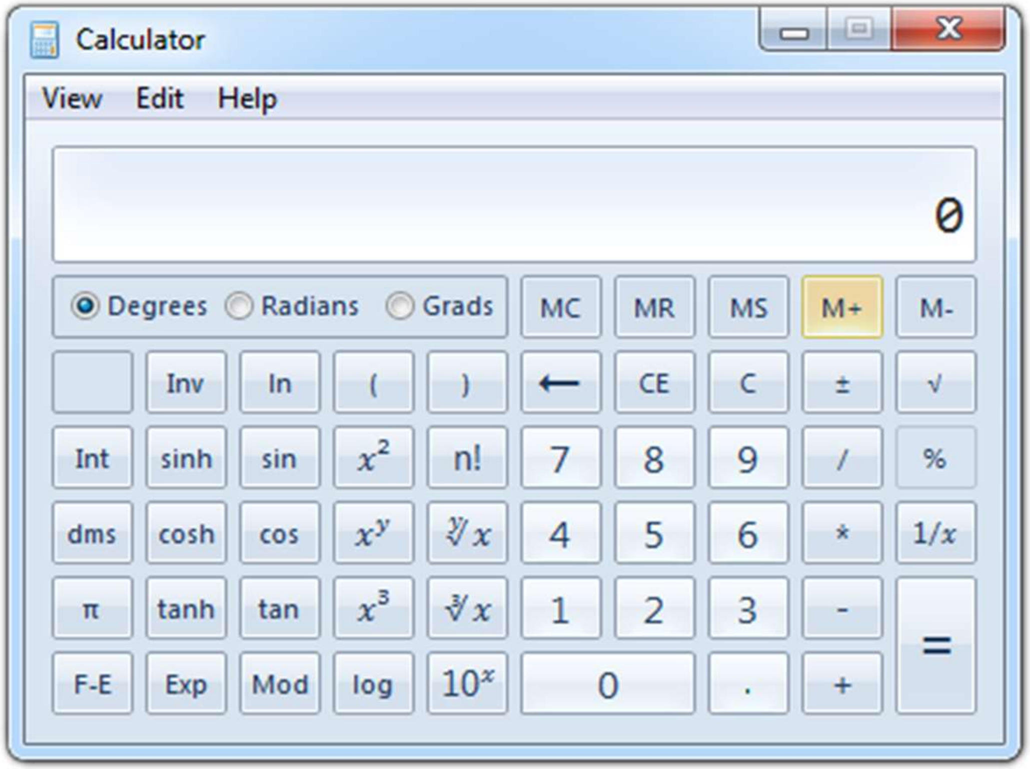
Class Hierarchies: Real World Example


Widget
abstract class
CourseViewsWidget
abstract class
Librarian Recommended Widget
E-Reserves Widget
Browse Subjects Widget
General Purpose Widgets


Vehicle
Car
Bicycle
Motorcycle
Bus
Truck
SUV

Real World Example: Calculator

- Creating an application like the Windows Calculator
  - Typical scenario for applying the object-oriented approach


Calculator
View Edit Help
0
Degrees Radians Grads
MC MR MS M+ M-
Inv ln ( ) ← CE C ± √
Int sinh sin x^2 n! 7 8 9 / %
dms cosh cos x^y $\sqrt[y]{x}$ 4 5 6 * 1/x
π tanh tan x^3 $\sqrt[3]{x}$ 1 2 3 - =
F-E Exp Mod log 10^x 0 . +

Real World Example: Calculator (2)

- The calculator consists of controls:
  - Buttons, panels, text boxes, menus, check boxes, radio buttons, etc.
- Class Control – the root of our OO hierarchy
  - All controls can be painted on the screen
    - Should implement an interface IPaintable with a method Paint()
  - Common properties: location, size, text, face color, font, background color, etc.

Real World Example: Calculator (3)

- Some controls could contain other (nested) controls inside (e. g. panels and toolbars)
  - We should have class Container that extends Control holding a collection of child controls

- The Calculator itself is a Form
  - Form is a special kind of Container
  - Contains also border, title (text derived from Control), icon and system buttons

- How the Calculator paints itself?
  - Invokes Paint() for all child controls inside it

Real World Example: Calculator (4)

- How a Container paints itself?
  - Invokes Paint() for all controls inside it

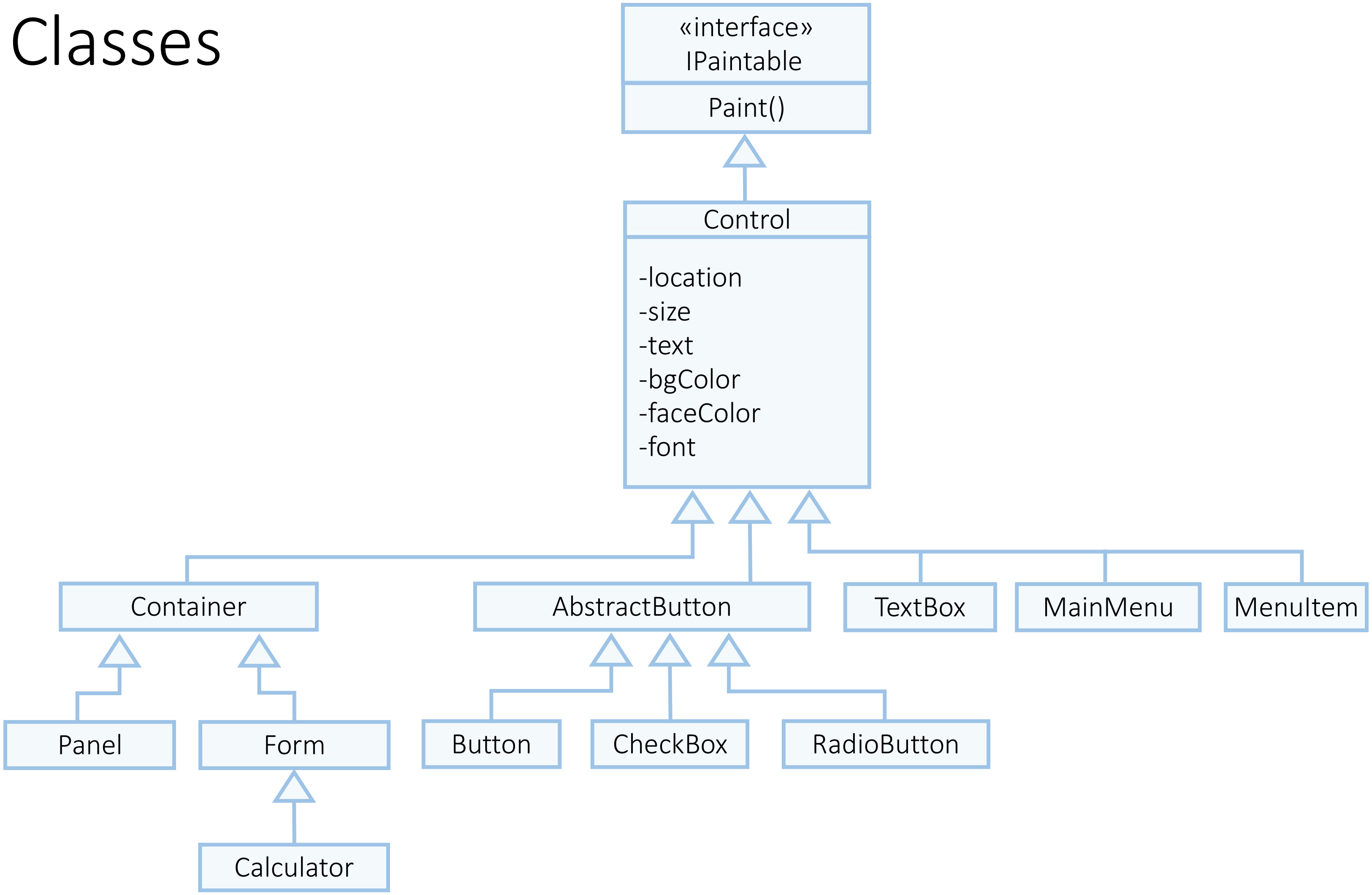
  - Each control knows how to visualize itself

- What is the common between buttons, check boxes and radio buttons?
  - Can be pressed

  - Can be selected

- We can define class AbstractButton and all buttons can derive from it

Calculator Classes


«interface»
IPaintable

Paint()

Control

-location
-size
-text
-bgColor
-faceColor
-font

Container

AbstractButton

TextBox

MainMenu

MenuItem

Panel

Form

Button

CheckBox

RadioButton

Calculator

Cohesion and Coupling

Cohesion

- Cohesion describes how closely all the routines in a class or all the code in a routine support a central purpose

- Cohesion must be strong

  - Well-defined abstractions keep cohesion strong

- Classes must contain strongly related functionality and aim for single purpose

- Cohesion is a useful tool for managing complexity

Good and Bad Cohesion

- Good: hard disk, cdrom, floppy

- BAD: spaghetti code

Strong Cohesion

- Strong cohesion example
  - Class Math that has methods:

    Sin(), Cos(), Asin()

    Sqrt(), Pow(), Exp()

    Math.PI, Math.E

```
double sideA = 40, sideB = 69;
double angleAB = Math.PI / 3;

double sideC =
    Math.Pow(sideA, 2) + Math.Pow(sideB, 2)
    - 2 * sideA * sideB * Math.Cos(angleAB);

double sidesSqrtSum = Math.Sqrt(sideA) +
Math.Sqrt(sideB) + Math.Sqrt(sideC);
```

Bad Cohesion

- Bad cohesion example
  - Class Magic that has these methods:

```
public void PrintDocument(Document d);
public void SendEmail(
  string recipient, string subject, string text);
public void CalculateDistanceBetweenPoints(
  int x1, int y1, int x2, int y2)
```

- Another example:

```
MagicClass.MakePizza("Fat Pepperoni");
MagicClass.WithdrawMoney("999e6");
MagicClass.OpenDBConnection();
```

Coupling

- Coupling describes how tightly a class or routine is related to other classes or routines

- Coupling must be kept loose

  - Modules must depend little on each other

  - All classes and routines must have small, direct, visible, and flexible relations to other classes and routines

  - One module must be easily used by other modules

Loose and Tight Coupling

- Loose Coupling:
  - Easily replace old HDD
  - Easily place this HDD to another motherboard

- Tight Coupling:
  - Where is the video adapter?
  - Can you change the video controller?

Loose Coupling – Example

```
class Report
{
    public bool LoadFromFile(string fileName) {…}
    public bool SaveToFile(string fileName) {…}
}
class Printer
{
    public static int Print(Report report) {…}
}
class Program
{
    static void Main()
    {
        Report myReport = new Report();
        myReport.LoadFromFile("C:\\DailyReport.rep");
        Printer.Print(myReport);
    }
}
```

Tight Coupling – Example

```
class MathParams
{
    public static double operand;
    public static double result;
}
class MathUtil
{
    public static void Sqrt()
    {
      MathParams.result = CalcSqrt(MathParams.operand);
    }
}
class MainClass
{
    static void Main()
    {
        MathParams.operand = 64;
        MathUtil.Sqrt();
        Console.WriteLine(MathParams.result);
    }
}
```

Spaghetti Code

- Combination of bad cohesion and tight coupling:

```
class Report
{
  public void Print() {…}
  public void InitPrinter() {…}
  public void LoadPrinterDriver(string fileName) {…}
  public bool SaveReport(string fileName) {…}
  public void SetPrinter(string printer) {…}
}
class Printer
{
  public void SetFileName() {…}
  public static bool LoadReport() {…}
  public static bool CheckReport() {…}
}
```

Summary

- OOP fundamental principals are: inheritance, encapsulation, abstraction, polymorphism
    - Inheritance allows inheriting members form another class
    - Abstraction and encapsulation hide internal data and allow working through abstract interface
    - Polymorphism allows working with objects through their parent interface and invoke abstract actions
- Strong cohesion and loose coupling avoid spaghetti code

After This Session You Should be comfortable with

- How to define classes, instantiate objects and access different part of an object
- What is a constructor and how is it defined and used
- What are accessor and mutator methods and how they can be used in conjunction with encapsulation
- What is method overloading and why is this regarded as good style
- How to represent a class using class diagrams (attributes, methods and access permissions) and the relationships between classes
- What is encapsulation/information-hiding, how is it done and why is it important to write programs that follow this principle

BITS Pilani
Pilani Campus

SOFTWARE DEVELOPMENT LIFECYCLE

Overview of OOAD and UP

Prof A R Rahman

Overview of OOAD

Object-Oriented Analysis and Design

Disclaimer

Slides come from a variety of sources: Craig Larman-developed slides; author of this classic textbook and other sources from the www

Agenda

- Object-Oriented Concepts and Principles
- Object Oriented Analysis
- Object Oriented Design
- Overview of Unified Process for software development

Introduction

We live in a world of objects.

They can be categorized, described, organized, combined, manipulated, and created.

Object-Oriented view is an abstraction that models the world in ways that help us to better understand and navigate it.

00 approach was first proposed in the late 1960s.

As time passes, object technologies are replacing classical software development approaches. Why?

The answer is not simple.

Object technologies lead to reuse, 00 software is easier to maintain, to adapt, and to scale.

Object-Oriented Paradigm

For many years, the term 00 was used to denote a software development approach that used one of a number of 00 programming languages e.g. Ada 95, C++, Eiffel, Smalltalk)

Today, the 00 paradigm encompasses a complete view of software engineering.

Software engineers and their managers must consider OORA, 00D, OODA(Dornan anal,), OODBMS.

Object-Oriented Concepts

- consider a chair.
- It is a member of a much larger class of objects that we call furniture.
- A set of generic attributes can be associated with every object in the class furniture (e.g. cost, dimensions, weight, location, color).
- Chair inherits all attributes defined for the class furniture.
- Every object in the class furniture can be bought, sold, modified, and moved.

Object-Oriented Concepts

- Each of these operations will modify one or more attributes of the object.
- The object Chair encapsulates data, operations, other objects (composite), constants, and other related information.
- Encapsulations means that all of this information is packaged under one name and can be reused as one specification or program component.
- Object-oriented = objects + classification + inheritance + communications.

Key Concepts

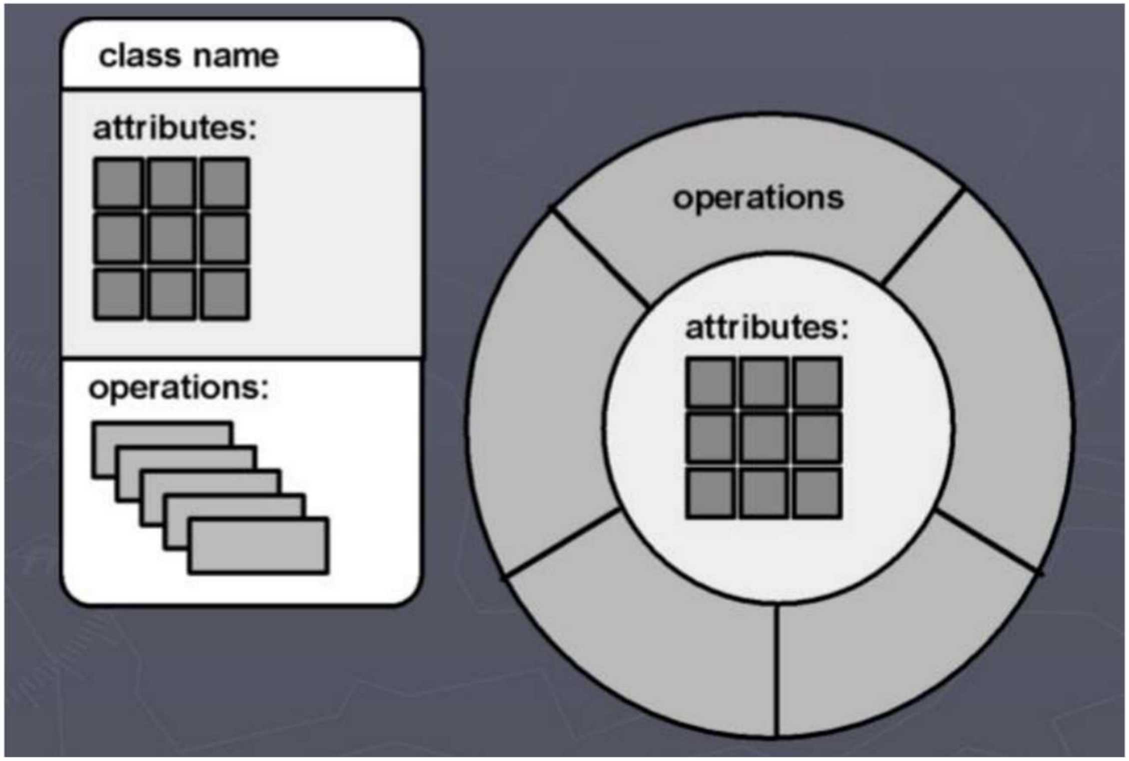
- classes and class hierarchies
- Instances
- Inheritance
- abstraction and hiding

- Objects
- Encapsulation
- polymorphism
- messages

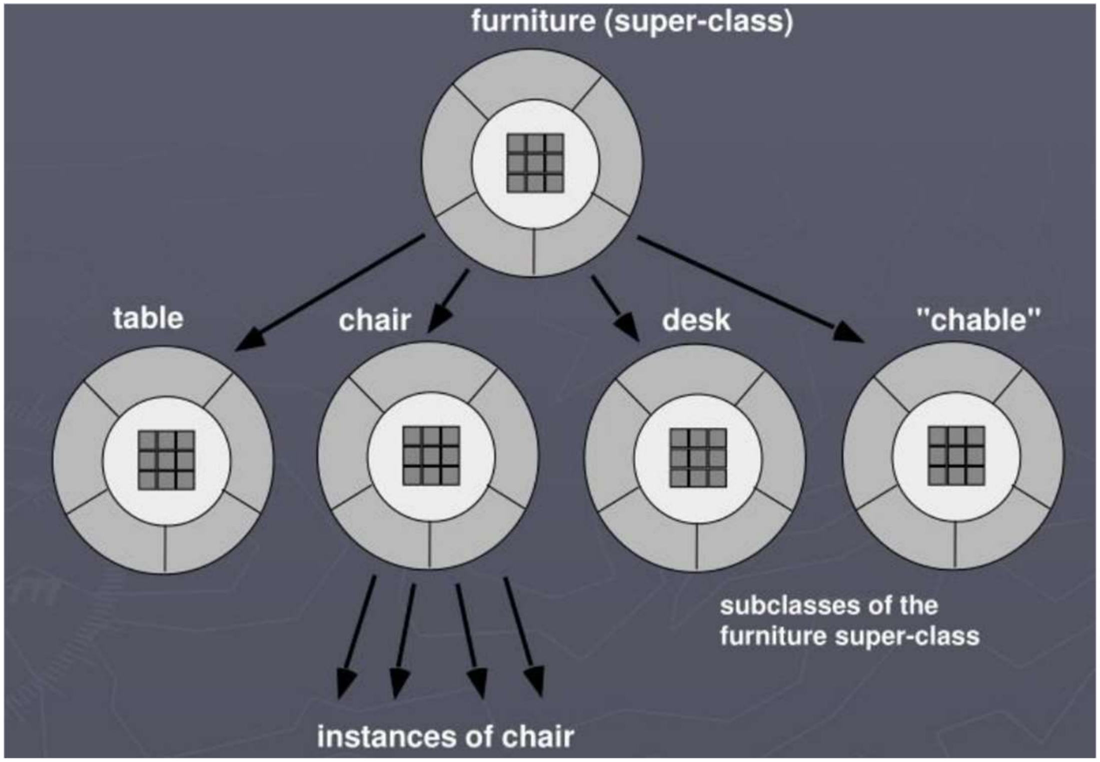
Classes and Objects

- A class is an OO concept that encapsulates the data and procedural abstractions that are required to describe the content and behavior of some real-world entity.
- The only way to reach the attributes is to go through one of the methods.
- This achieves information hiding and reduces the impact of side effects associated with changes.
- The methods are cohesive and the class tends to be decoupled from other elements of a system.
- All of these design characteristics lead to high-quality software.

Building a Class


class name
attributes:
operations:
operations
attributes:

Class Hierarchy


furniture (super-class)
table
chair
desk
"chable"
subclasses of the furniture super-class
instances of chair

Messages

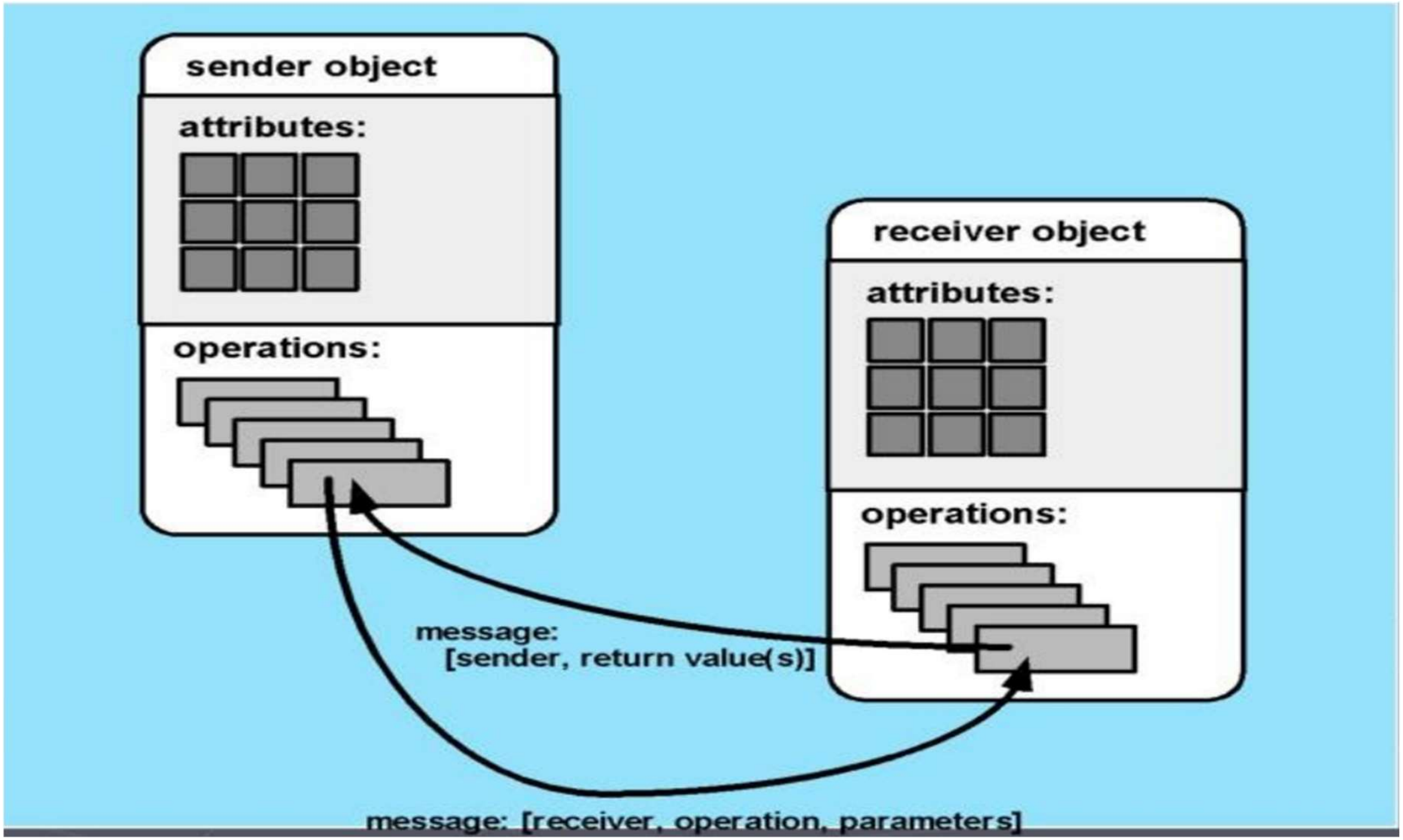
- Messages are the means by which objects interact.

- The behavior is accomplished when an operation is executed.

- An operation within a sender object generates a message of the form:
    - [destination, operation, parameters]

- The receiver object responds to the message by first choosing the operation implements the message, executing this operation and then returning control to the caller.

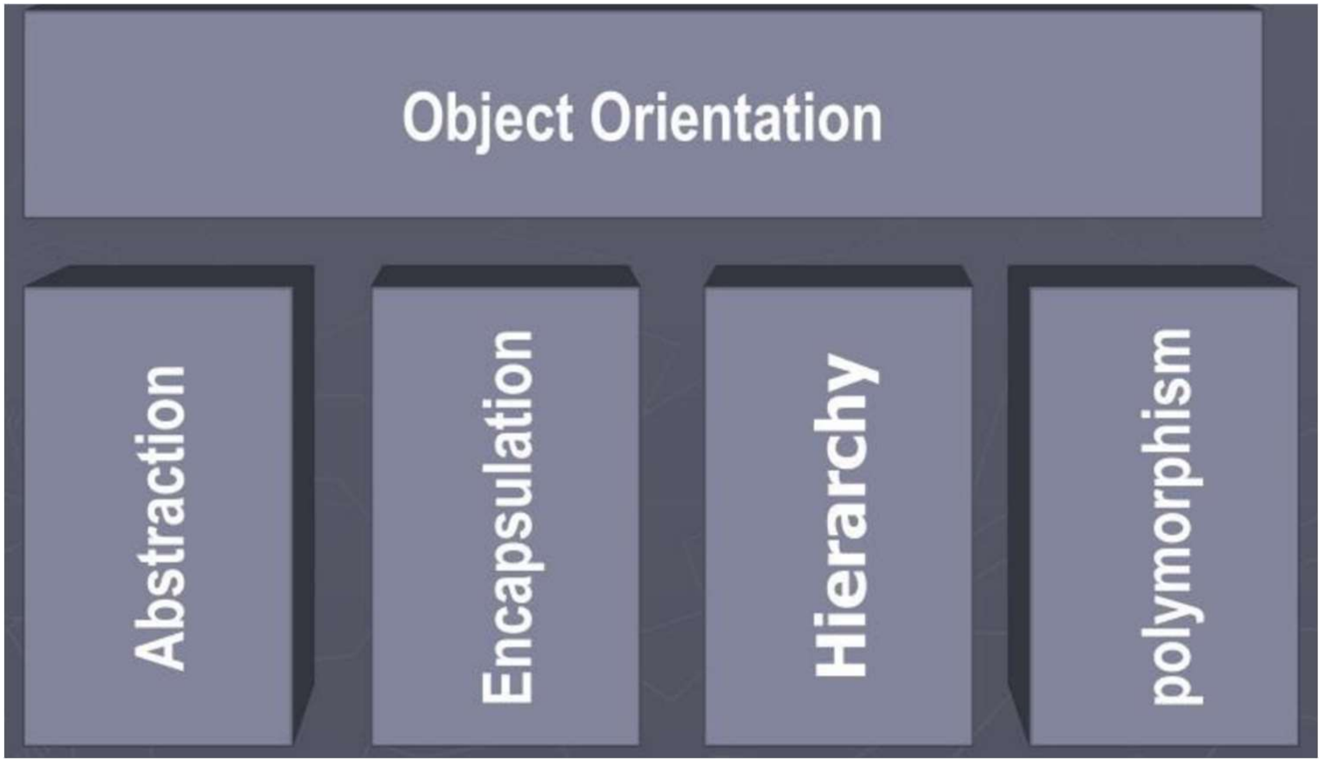
Messages


sender object
attributes:
operations:
receiver object
attributes:
operations:
message:
[sender, return value(s)]
message: [receiver, operation, parameters]

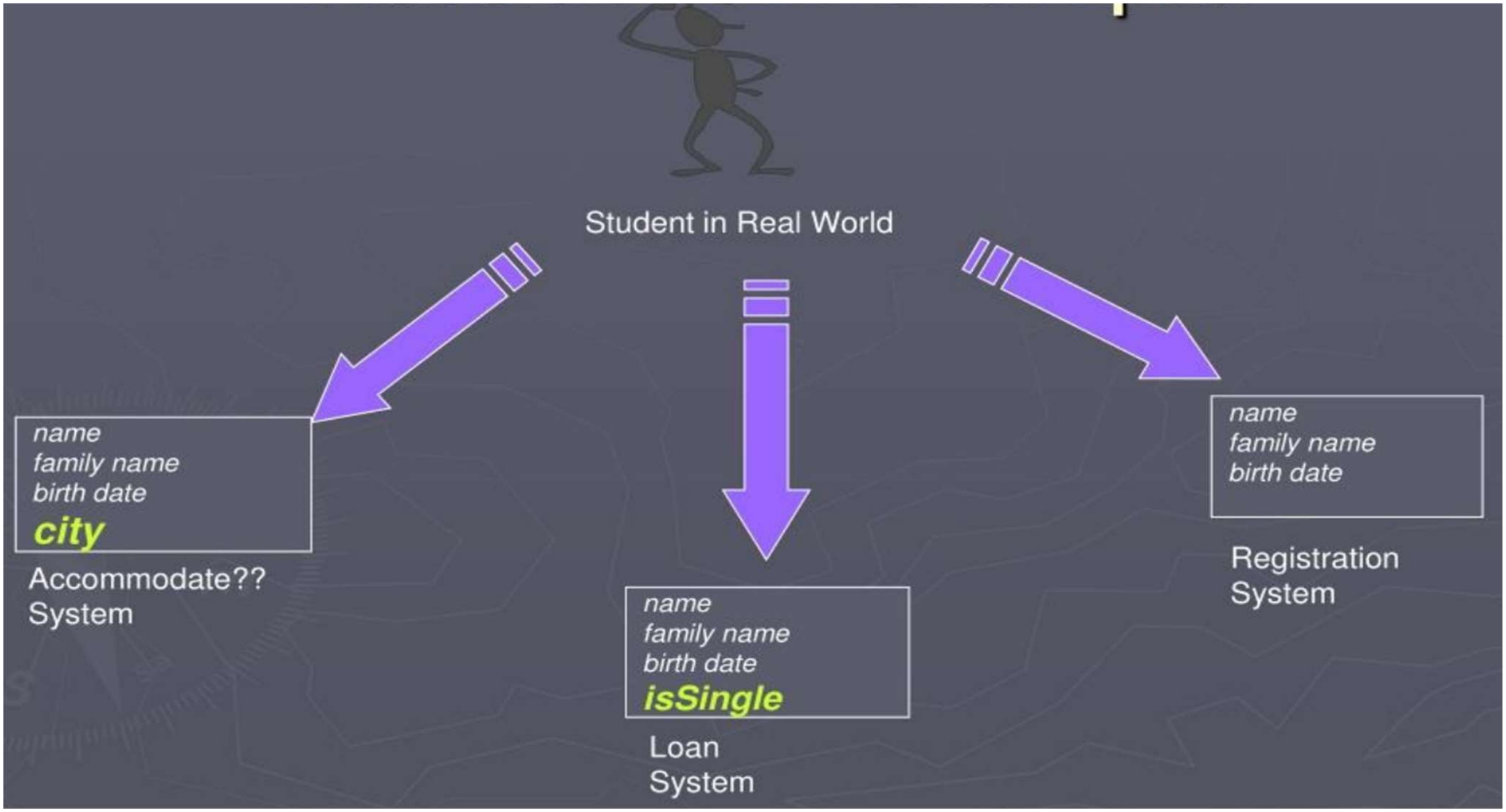
Polymorphism

- It is a characteristic that greatly reduces the effort required to extend an existing 00 system.
- Polymorphism enables a number of different operations to have the same name.
- For example: drawing different type of graphs (e.g. Line, Pie, Histogram)
- It decouples objects from one another making each more independent.
- General class graph and one subclass for each type.

Basic Principles of Object Orientation


Object Orientation
Abstraction
Encapsulation
Hierarchy
polymorphism

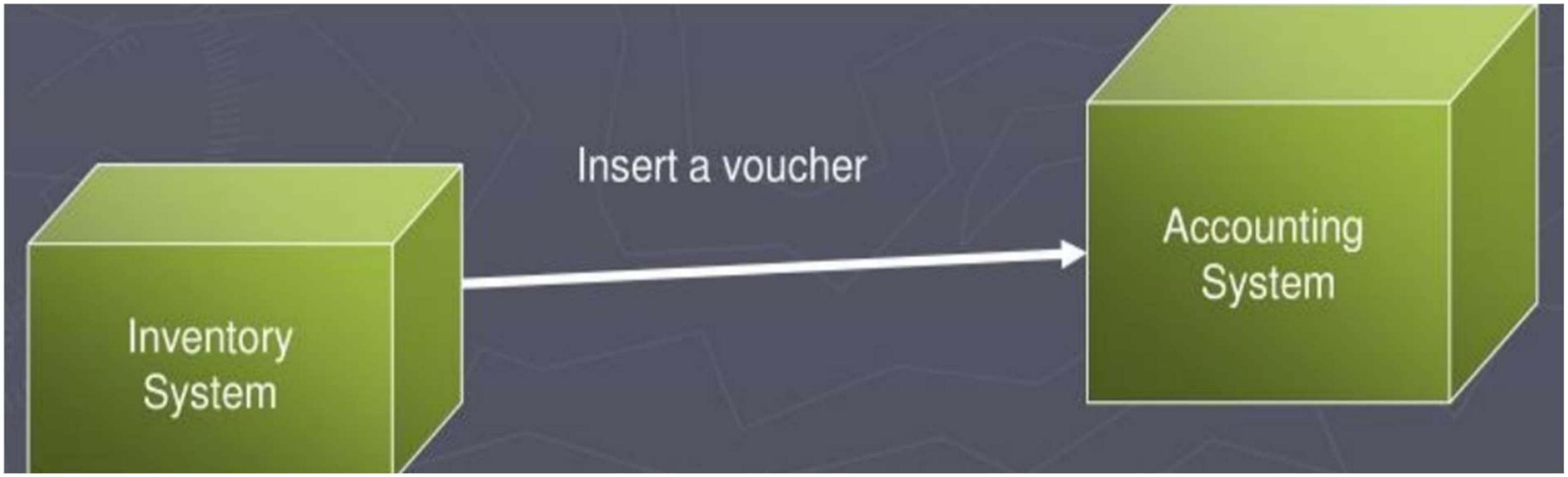
Abstraction: example


Student in Real World
name
family name
birth date
city
Accommodate??
System
name
family name
birth date
isSingle
Loan
System
name
family name
birth date
Registration
System

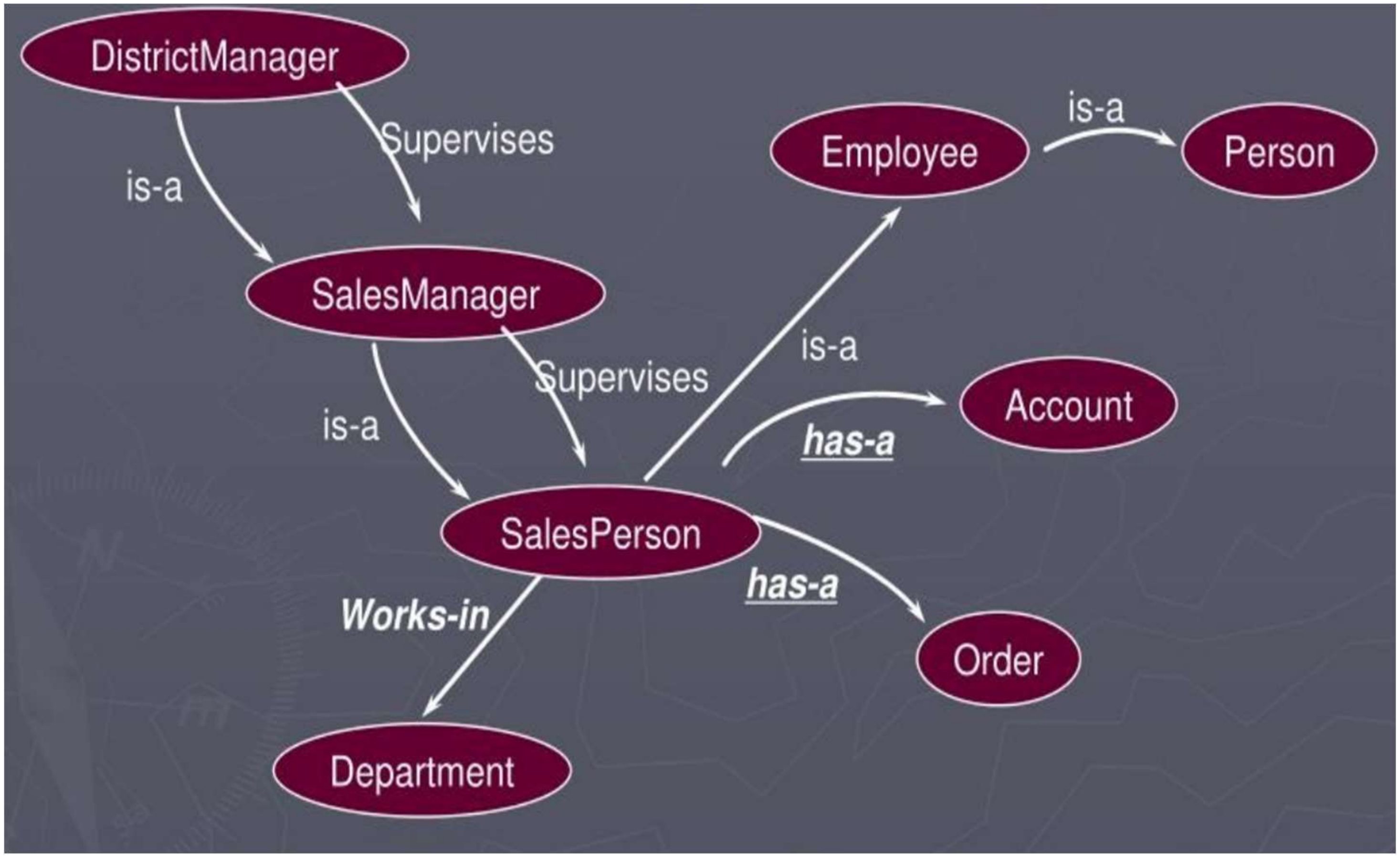
Encapsulation: example

- Inventory System know how to insert a voucher in Accounting System and not more.

- How a voucher insert in Accounting System is private

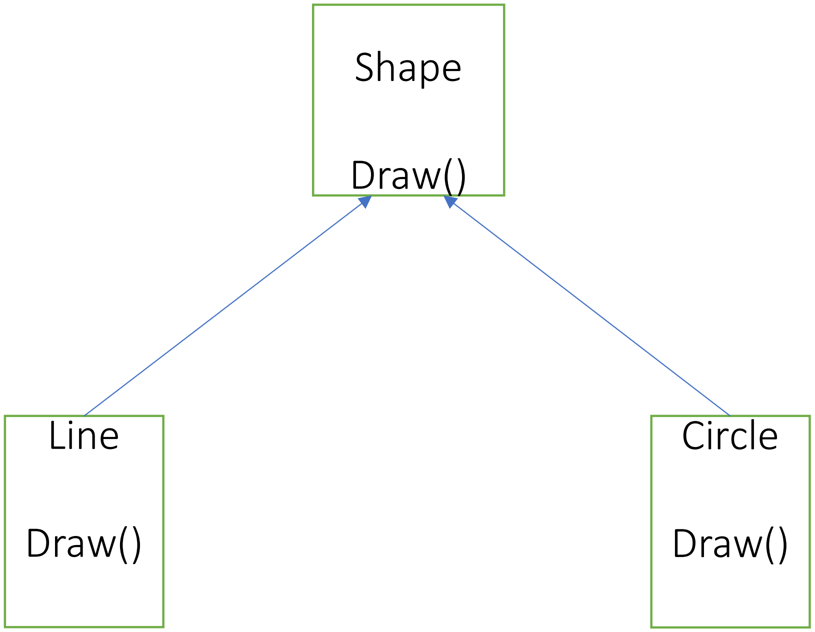

Insert a voucher
Inventory System
Accounting System

Hierarchy: Example


DistrictManager
is-a
Supervises
SalesManager
is-a
Supervises
SalesPerson
Works-in
Department
is-a
Employee
is-a
Person
has-a
Account
has-a
Order

A Semantic Network for SalesPerson

Polymorphism


Shape

Draw()

Line

Draw()

Circle

Draw()

Object-Oriented Analysis

Introduction

- Object oriented analysis is based upon concept that we first learned in kindergarten: objects and attributes, classes and members, wholes and parts.

- To define all classes, the operations, the attributes, the relationships and the behavior the following tasks must be done:

- Basic user requirements must be communicated between the customer and the software engineer.

- Classes must be identified.

- A class hierarchy must be specified.

Introduction

- Object to object relationships should be represented.
- Object behavior must be modeled.
- Above Tasks must be reapplied iteratively until the model is complete.
- There is no universal agreement on the concepts that serve as a foundation for 00A.

Conventional vs. OO Approaches

- Structured Analysis (SA):
- takes a distinct input-process-output view of requirements.
- Data are considered separately from the processes that transform the data.
- System behavior tends to play a secondary role.
- makes heavy use of functional decomposition.

Modeling Dimensions

1. Identification/classification of entities.
2. General-to-specific and whole-to-part entity relationships
3. Other entity relationships
4. Description of attributes of entities
5. Large-scale model partitioning
6. States and transitions between states
7. Detailed specification for functions
8. Top-down decomposition
9. End-to-end processing sequences
10. Identification of exclusive services
11. Entity communications (via messages or events)

The modeling dimensions 8 and 9 are always present with SA.

The 00A Landscape

- Dozens of 00A method during the late 1980s and into the 1990s are introduced.
- Each of them proposed:
- A process for the analysis of the product or system
- A set of diagrams that evolved out of the process.
- A notation that enabled the software engineer to create the analysis model in a consistent manner.

The 00A Landscape

- The most widely use were:
- The Booch method (an evolutionary approach is maintained).
- The Rumbaugh method (Object modeling technique (0MT))
- The Jacobson method (00 Software Engineering (00SE))
- The Coad and Yourdon method (One of the easiest)
- The Wirfs-Brook method (do not make clear distinction between design and analysis tasks)

Generic Steps for 00A

- Elicit customer requirements for the system.

- Identify scenarios for use-cases.

- Select classes and objects using basic requirements as a guide.

- Identify attributes and operations for each system object.

- Define structures and hierarchies that organize classes.
- Build an object-behavior model.
- Review the 00 analysis model against use-cases or scenarios.

A Unified Approach to 00A

- Grady Booch, James Rumbaugh and Ivar Jacobson combine the best features into a unified method called: Unified Modeling Language (UML)
- UML allows a software engineer to express and analysis model using a modeling notation that is governed by a set of Syntactic, Semantic, and Pragmatic rules.
- The syntax tells us how the symbols should look and how they are combined. (Word in natural language)
- The semantics tells us what each symbols means and how it should be interpreted. (Meaning of words in natural language)
- The pragmatic rules define the intentions of the symbols through which the purpose of the model is achieved and become understandable.
  (The rules for constructing sentences that are clear and understandable in natural language)

Unified Modeling Language (UML)

- User model view. This view represents the system (product) from the user's called "actors" in UML perspective

- Structural model view. Data and functionality in viewed from inside the system That is, static structure (Classes, Objects and relationships) is modelled

- Behaviour model view. This part of the analysis model represents the dynamic or behavioral aspects of the system.

- Implementation model view. The structural and behavioral aspects of the system are represented as they are to be built.

- Environment model view. The structural and behavioral aspects of the environment in which the system is to be implemented are represented.

UML analysis modeling focuses on the first two views of the system.

UML design modeling addresses the other three views.

The 00A Process

- The 00A process begins with an understanding of the manner in which the system will be used by:
- People, if the system is human-interactive.
- Machines, if the system is involved in process control.
- Programs, if the system coordinates and controls applications
- Once the scenario of usage has been defined, the modeling of the software begins.
- A series of techniques may be used to gather basic customer requirements.

What is UML :

UML (Unified Modeling Language)
A language for

- visualizing
- specifying
- constructing
- documenting

the artifacts of a software-intensive system.

Diagrams in UML :

- Use case diagram
- Class diagram
- Object diagram
- Sequence diagram
- Communication/Collaboration diagram
- Statechart diagram
- Activity diagram
- Component diagram
- Deployment diagram

Major Diagrams in UML :

- Use case diagrams
  - Describe the functional behavior of the system as seen by the user
- Class diagrams
  - Describe the static structure of the system: Objects, attributes, associations
- Sequence diagrams
  - Describe the dynamic behavior between objects of the system
- Statechart diagrams
  - Describe the dynamic behavior of an individual object
- Activity diagrams
  - Describe the dynamic behavior of a system, in particular the workflow.

Thinking in Objects and UML - 1

- The Unified Modeling Language (UML) is a standard diagramming notation; sometimes referred to as a blueprint.
- It is NOT OOA/OOD or a method
- Only a notation for capturing objects and the relationships among objects (dependency; inheritance; realizes; aggregates, . .)
- UML is language-independent
- Analysis and design provide software "blueprints" captured in UML.
- Blueprints serve as a tool for thought and as a form of communication with others.

Thinking in Objects and UML – 2

- But it is far more essential to 'think' in terms of objects as providing 'services' and accommodating 'responsibilities.'

- Discuss: What is meant by 'services?'
- How indicated?
  - How might you think these 'services' impact the design of classes?
  - How might a client access these services?
  - What is encapsulation?
  - How does it relate to reusability?
  - Self-governance?
  - Design?
- Discuss: What is meant by 'responsibilities?'
  - Encapsulation of data and services?

Thinking in Terms of Objects and UML - 3

- Object-Oriented Analysis (Overview)
  - An investigation of the problem (rather than how a solution is defined)
  - During OO analysis, there is an emphasis on finding and describing the objects (or concepts) in the problem domain.
  - For example, concepts in a Library Information System include *Book*, and *Library*.
  - High level views found in the application domain.
  - Often, sometimes called domain objects; entities.

Object-Oriented Design

Introduction

OOD transforms the analysis model created using 00A into a design model that serves as a blueprint for software construction.

OOD results in a design that achieves a number of different levels of modularity.

Subsystems: Major system components.
Objects: Data and the operations.

- Four important software design concepts:
  Abstraction
  Information Hiding
  Functional Independence
  Modularity

OOD provides a mechanism that enables the designer to achieve all four with less complexity and compromise.

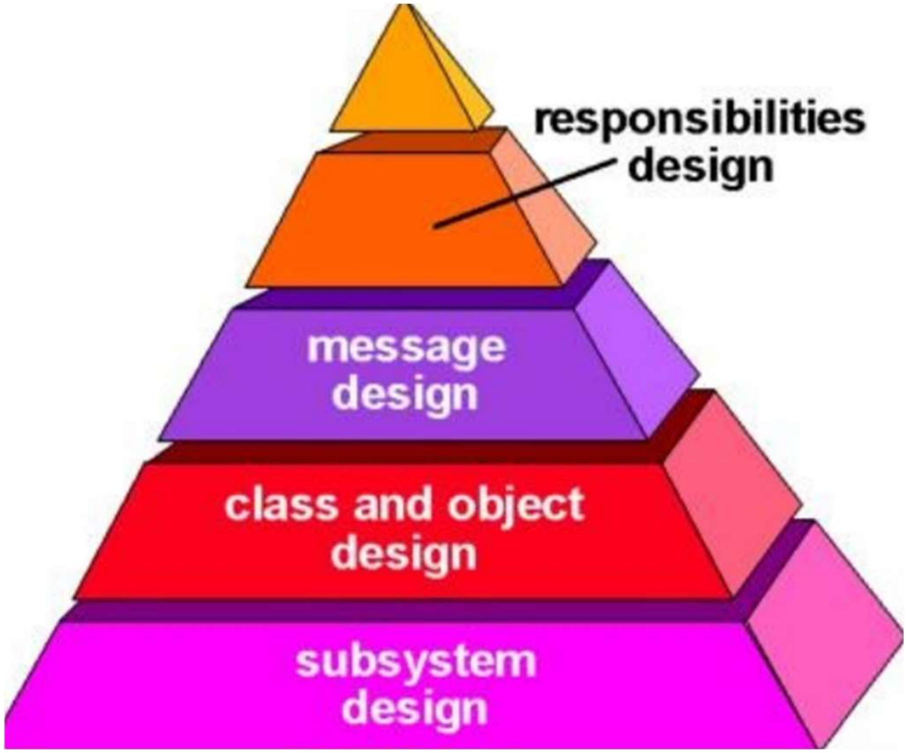
Object-Oriented Design

The subsystem layer: Representation of each of the subsystems that enable the software to achieve its customer defined requirements

The class and object layer: The class hierarchies,(generalization) and representation of objects.

The message layer: The design details of communication of each object with its collaborators.(external and internal interfaces)

The responsibilities layer: Data Structure and algorithmic design for all attributes and operations.


responsibilities design
message design
class and object design
subsystem design

Object-Oriented Design

The design pyramid focuses exclusively on the design of a specific product or system.

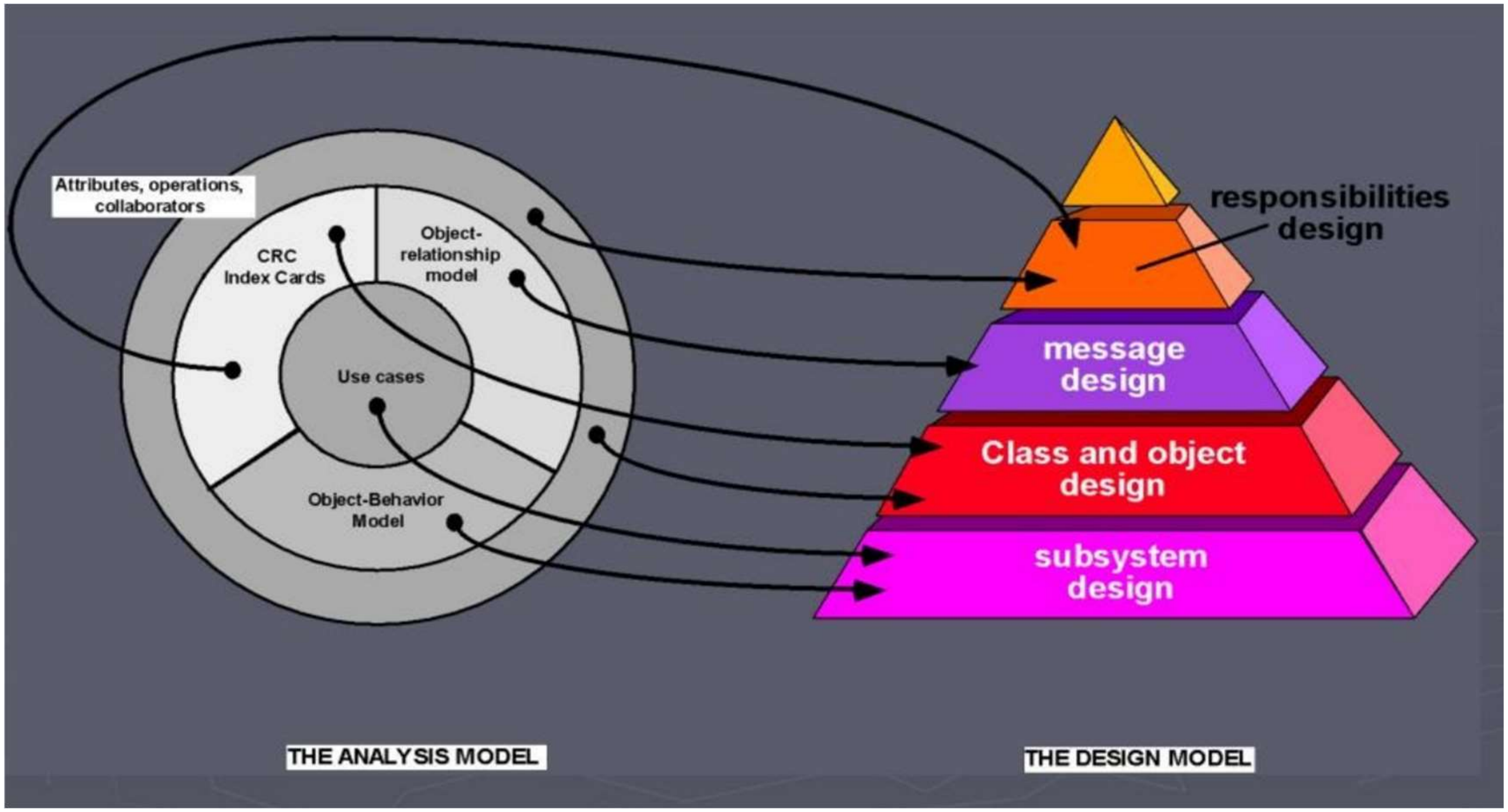
Another layer of design which forms the foundation on which the pyramid rests, exists.

The foundation layer focuses on the design of domain objects.

Domain objects play a key role in building the infrastructure for the OO system by providing support for:

Human/computer interface activities,
Task management,
Data management.

OOA and OOD


Attributes, operations, collaborators
CRC Index Cards
Object-relationship model
Use cases
Object-Behavior Model
responsibilities design
message design
Class and object design
subsystem design
THE ANALYSIS MODEL
THE DESIGN MODEL

Generic Components for OOD

Problem domain component the subsystems that are responsible for implementing customer requirements directly;

Human interaction component —the subsystems that implement the user interface (this included reusable GUI subsystems);

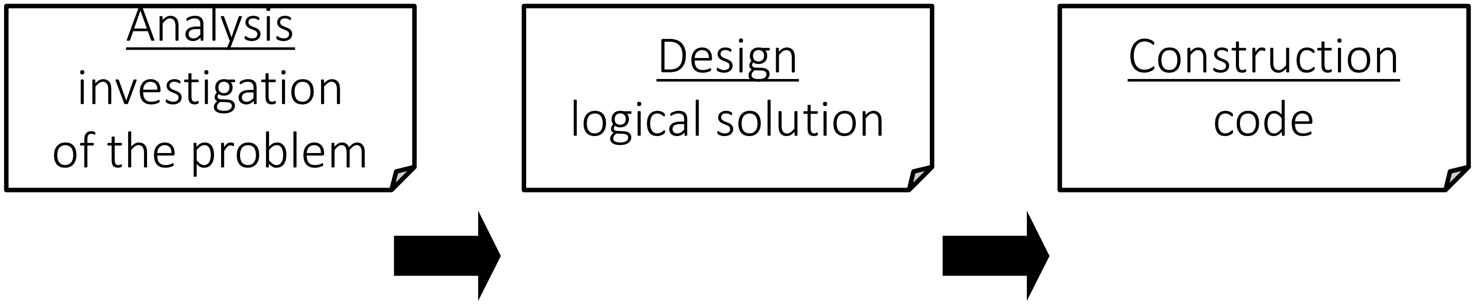
Task Management Component—the subsystems that are responsible for controlling and coordinating concurrent tasks that may be packaged within a subsystem or among different subsystems;

Data management component the subsystem that is responsible for the storage and retrieval of objects.

Thinking in Terms of Objects and UML - 4

- Object-Oriented Design
  - Emphasizes a conceptual solution that fulfills the requirements.
  - Need to define software objects and how they collaborate to meet the requirements.
  - For example, in the Library Information System, a *Book* software object may have a *title* attribute and a *getChapter* method.
    - What are the methods needed to process the attributes?
- Designs are implemented in a programming language.
  - In the example, we will have a *Book* class in Java.

Thinking in Terms of Objects and UML – 5
From Design to Implementation

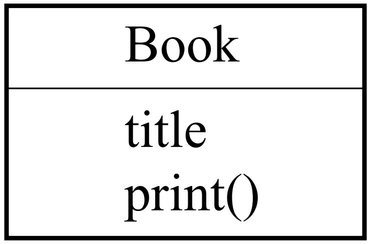


Analysis
investigation
of the problem

Design
logical solution

Construction
code

Book
(concept)



```
public class Book {
      public void print();
      private String title;
}
```

Domain concept

Representation in analysis of concepts

Representation in an object-oriented programming language.

Can you see the services / responsibilities in the Book class?

Thinking in Objects and UML-6

- Then too, there are sets of **proven design solutions** to problems that are considered 'best practices.'
  - Certain 'groupings' of classes with specific responsibilities / interfaces.
  - These provide specific solutions to specific problems.
  - Called **Design Patterns**
- We will discuss (much later) these standard patterns and how to **apply** them to develop solutions to common design problems.

Thinking in Objects and UML-7

- Of course, design (solution to requirements) ‘assume’ a robust **requirements analysis** has taken place.

- **Use Cases** are often used to capture **stories** of requirements and are often views as ‘constituting’ the **functional** requirements, but NOT the software quality factors (non-functional requirements).

- Use Cases are **not specifically designed** to be object-oriented, but rather are meant to capture how an application will be used.

- Many methods for capturing requirements.

- We will concentrate on Use Cases (Soon).

Object Oriented
Analysis and Design Using
UML

Evolution of Object-Oriented Development Methods

- Mid to late 1980s
  - Object-Oriented Languages (esp. C++) were very much in vogue
  - However, there was little guidance on how to divide a problem into OO classes.

UML

- Unified Modeling Language

  - In early 90s, there were many competing graphical notations all used for OOA.

  - Three of the major players got together in Booch’s company (The Three Amigos)

    - Rational Software Corporation
      - Grady Booch, James Rumbaugh, Ivor Jacobson

    - Merged their ideas to produce
      - UML (public domain)
      - Associated tools (mainly Rational Rose)
      - Rational Software Process (public domain)
      - Acquired other companies (Purify, Quantify, …)

Uses for UML

- OOA
  - A visual language for, in the problem domain,
    - capturing knowledge about a subject
    - expressing knowledge for the purposes of communication

- OOD
  - A visual language for, in the solution space
    - capturing design ideas
    - communicating design ideas

- Related, but distinct usages

- Must supplement both with written explanations

- UML
  - Has its warts
  - Good enough when augmented by written explanation

- Cover only the most useful subset of UML
  - Mainly class/object/use case/sequence charts.

- OMG-endorsed standard (Object Management Group)
  - UML Semantics Document
    - “inside-view”
    - specifies semantics of constructs

  - UML Notation Guide
    - “outside-view”
    - specifies notation for expressing constructs

UML is For

- For Problems
  - Specifying
  - Visualizing
  - Promoting Understanding
  - Documenting

- For Solutions
  - Specifying
  - Visualizing
  - Evaluating
  - Constructing
  - Documenting

UML is For

- Object Constraint Language specification document
  - definition of a (textual) language for expressing logical constraints

For Problem Solving

- Capturing Attempts
- Communicating Attempts
- Leveraging Knowledge

Parts of UML

- Class Diagrams
  - Models

- Object Diagrams
  - example models

- Use Case Diagrams
  - document who can do what in a system

- Sequence Diagrams
  - shows interactions between objects used to implement a use case

- Collaboration Diagrams
  - same as above, different style

Parts of UML (Cont…)

- State chart Diagrams
    - possible states and responses of a class and what transitions them

- Activity Diagrams
    - describe the behaviour of a class in response to internal processing

- Component Diagrams
    - Organization of and dependencies amongst software implementation components

- Deployment Diagrams
    - Describe the mapping of software implementation components onto processing nodes

- The real world is impenetrably complex
  - e.g., a complete model of you would include DNA, behaviour specifications, total history, parents’ history, influences, …
  - for a particular problem, abstracting you as
    - last name
    - first name
    - student number
    - course
    - final grade

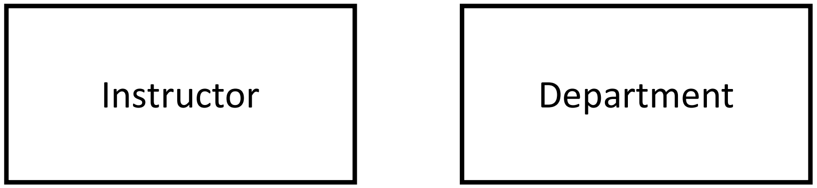
- An object is
  A concept, abstraction, or thing with crisp boundaries and meaning for the problem at hand

- Objects
  - promote understanding of the real world
  - provide a practical basis for computer implementation

- Decomposition of a problem into objects depends on
  - Judgment
  - The nature of the problem being solved
    - Not only the domain: two analyses of the same domain will turn out differently depending upon the kind of programs we wish to produce.

Classes

- A class describes a group of objects with similar properties.
    - *Class*: Instructor
        - *Object*: AR
        - *Object*: John



Instructor

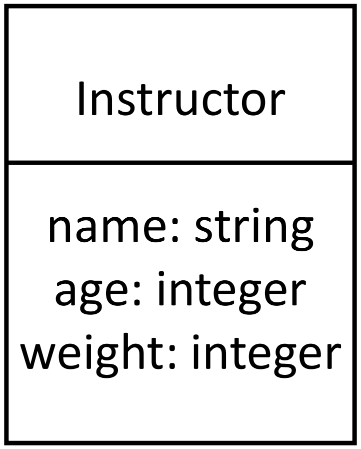
Department

Class: Department

Object: Department of Computer Science
Object: Department of Electrical Engineering

Attributes

- Data values held by the objects of a class

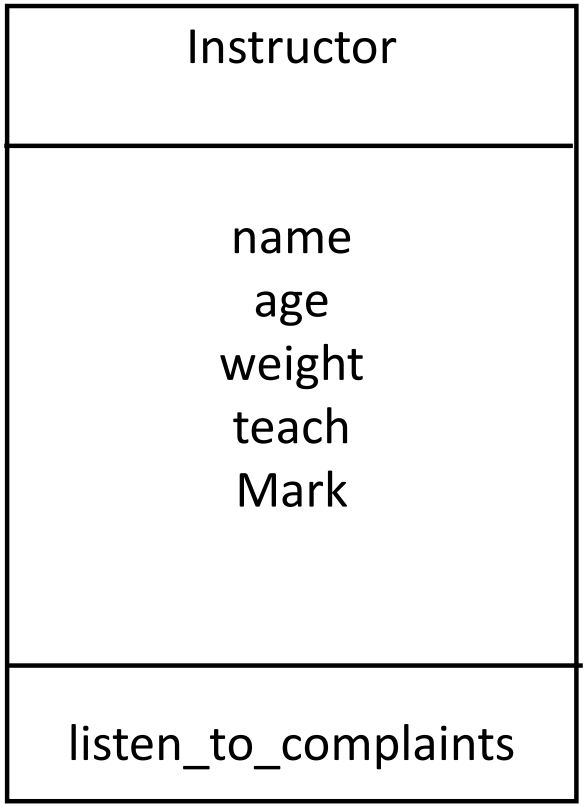


Instructor

name: string
age: integer
weight: integer

Operations

- A function or a transformation that may be applied to or by objects in a class.
  - Not often used (not often terribly useful) in an OOA



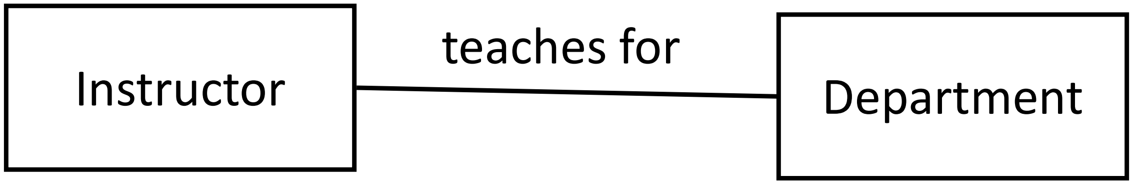
Instructor

name
age
weight
teach
Mark

listen_to_complaints

Links and Associations

- The means for establishing relationships among objects and classes.
  - link: a connection between two object instances
  - association: a collection of links with common structure and semantics.


Instructor
teaches for
Department

Object Diagrams

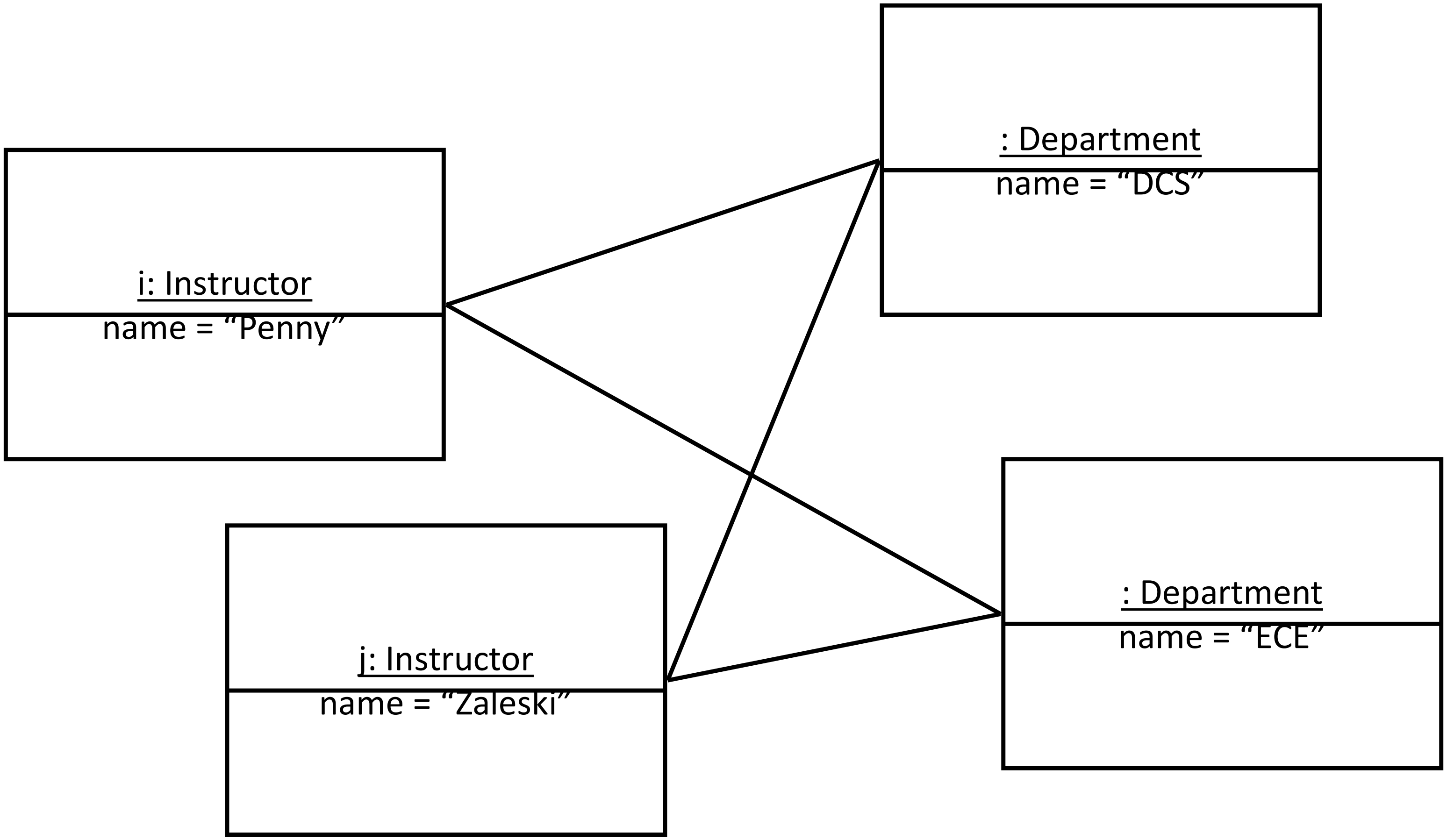
- Models instances of things contained in class diagrams.

- Shows a set of objects and their links at a point in time

- Useful preparatory to deciding on class structures.

- Useful in order to better explain more complex class diagrams by giving instance examples.

Object Diagrams



i: Instructor
name = “Penny”

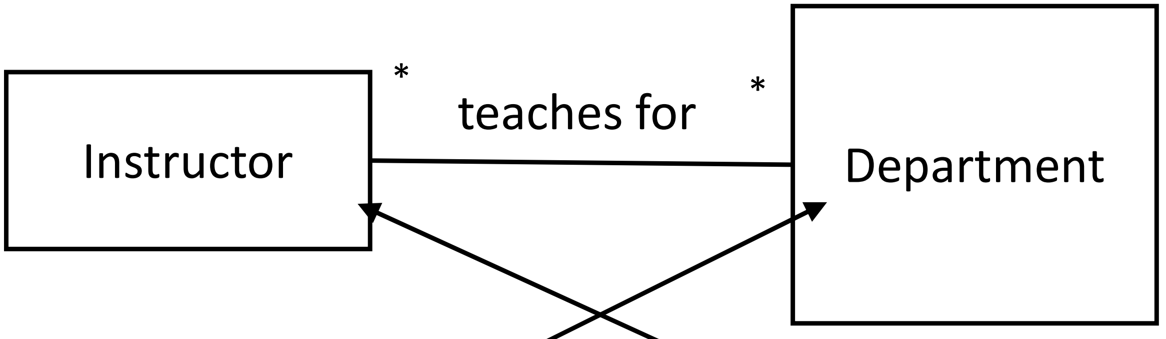
j: Instructor
name = “Zaleski”

: Department
name = “DCS”

: Department
name = “ECE”

Multiplicity

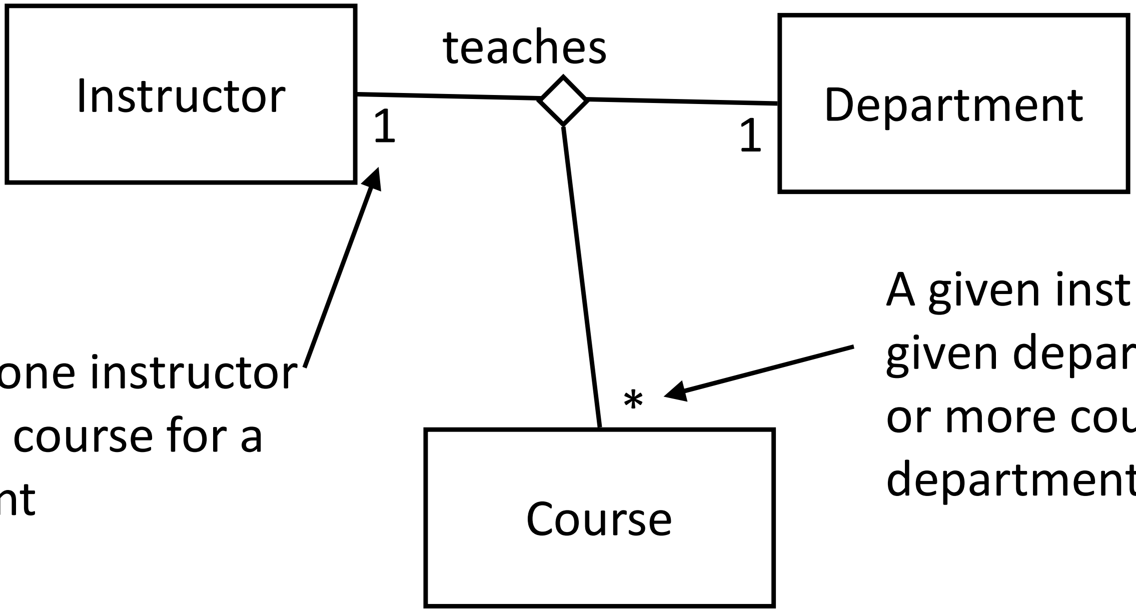
- Used to indicate the number of potential instances involved in the association when the other associated classes are fixed.



A given instructor can teach for potentially many departments (or none)

A given department employs zero or more instructors

N-ary Associations

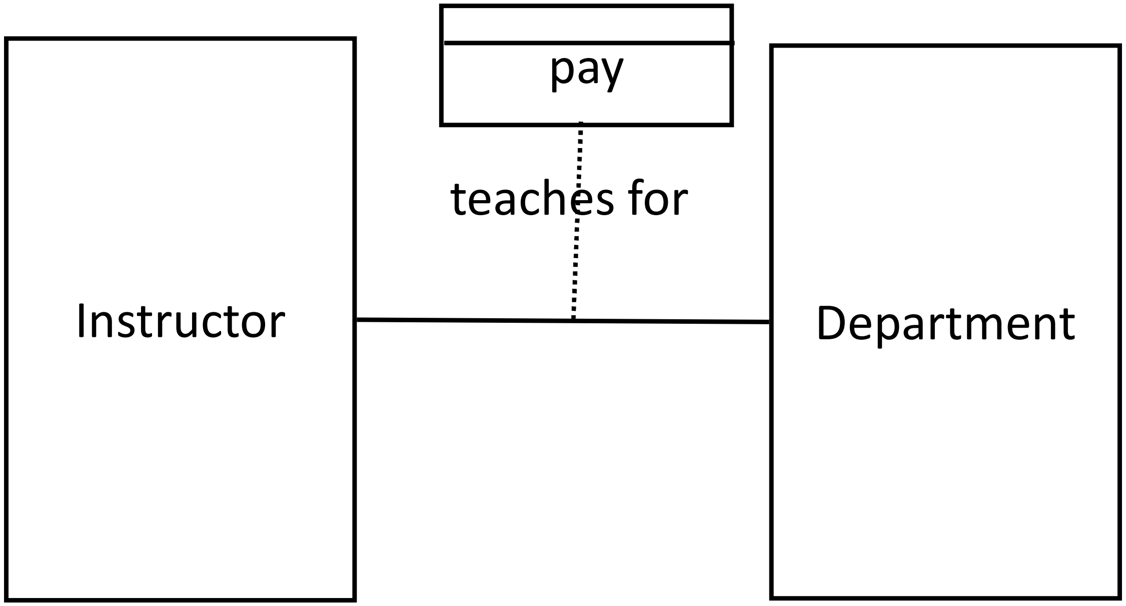

Instructor
teaches
Department
1
1
*
Course

There is exactly one instructor teaching a given course for a given department

A given instructor teaching for a given department may teach zero or more courses for that department.

Try to avoid them!

Attributes on Associations

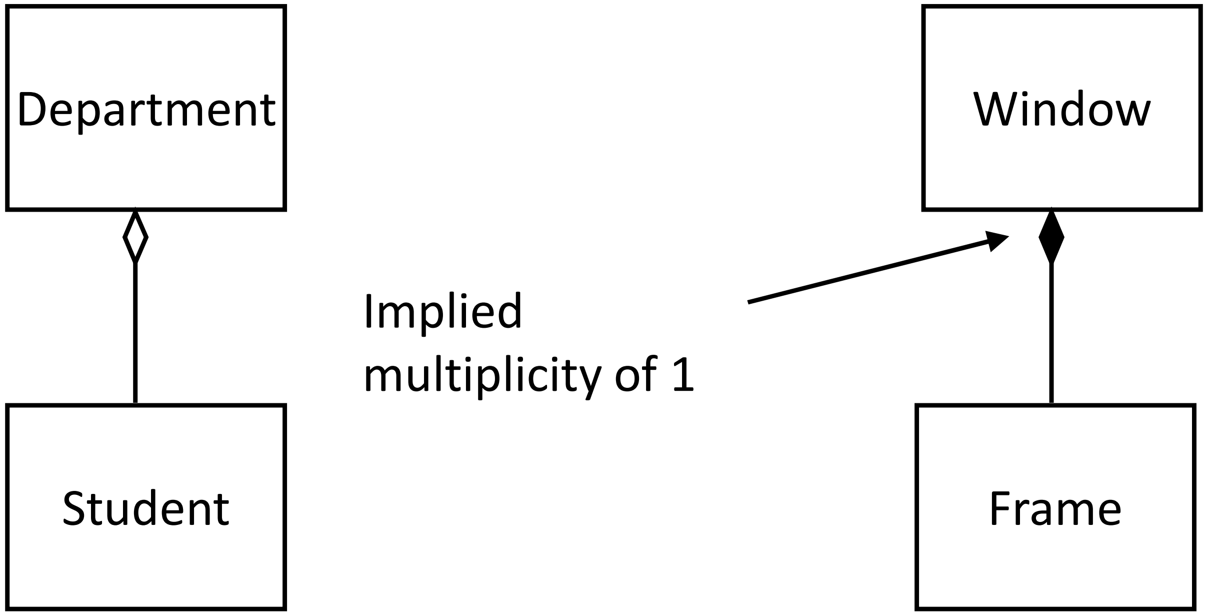

pay

teaches for

Instructor

Department

Aggregation Indicators (Part-Of)

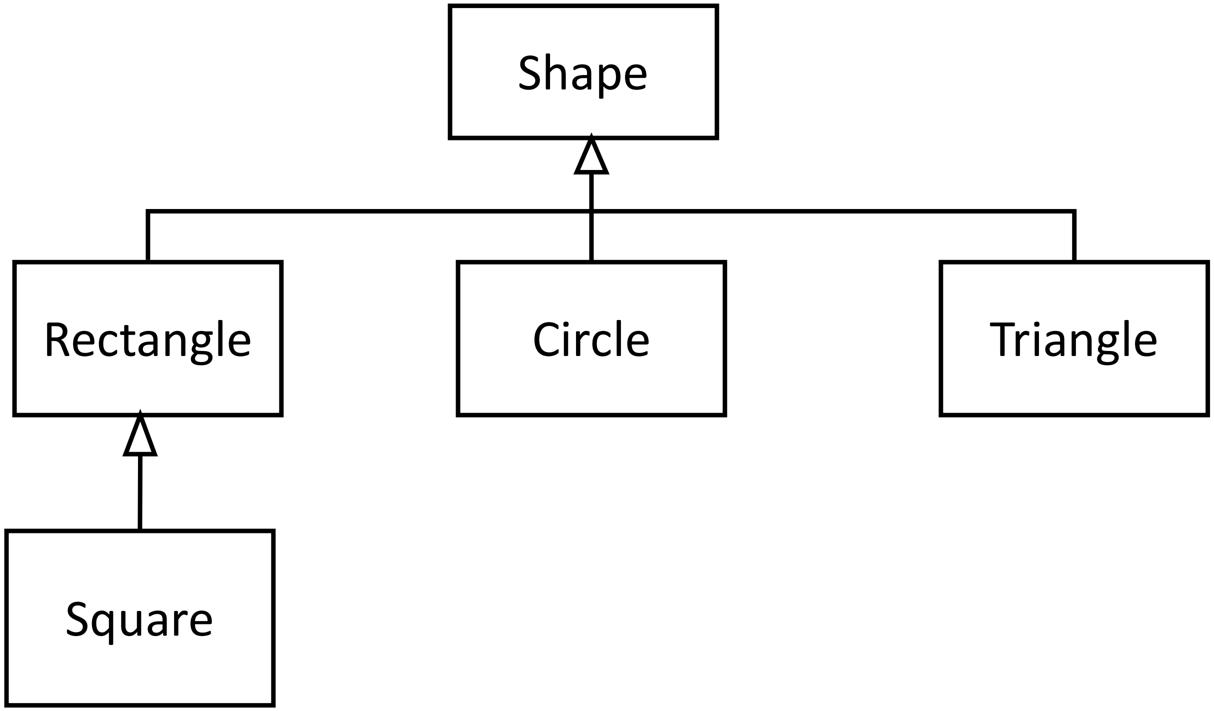

Department
Student
Window
Frame

Implied
multiplicity of 1

Aggregation
(no associated semantics)

Composition
(strong ownership,
coincident lifetime)

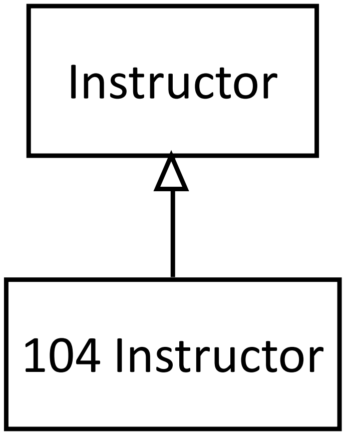
Generalization (a.k.a. Inheritance, is-a)


Shape
Rectangle
Circle
Triangle
Square

Avoiding Morphing Classes

- Analysis shown below may not be a good choice, as objects of class 104 Instructor may teach other things and different things next term.

- Avoid situations where objects will need to morph classes



Example

- We are asked to build a system for keeping track of the time our workers spend working on customer projects.

- We divide projects into activities, and the activities into tasks.

- A task is assigned to a worker, who could be a salaried worker or an hourly worker.

- Each task requires a certain skill, and resources have various skills at various level of expertise.

Steps

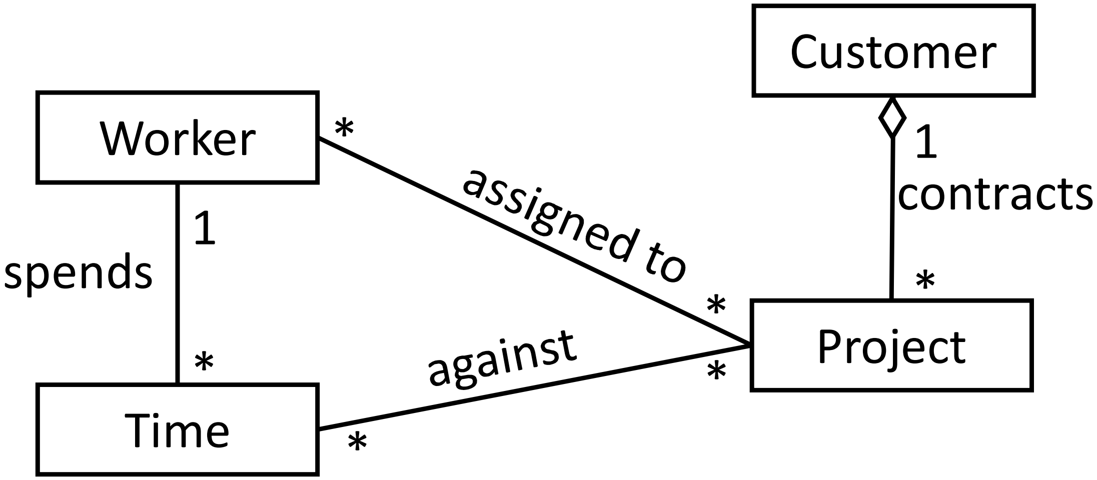
- Analyze the written requirements
  - Extract nouns: make them classes
  - Extract verbs: make them associations
  - Draw the OOA UML class diagrams
  - Determine attributes
  - Draw object diagrams to clarify class diagrams

Steps (continued..)

- Determine the system’s use cases
  - Identify Actors
  - Identify use case
  - Relate use cases
- Draw sequence diagrams
  - One per use case
  - Use to assign responsibilities to classes
- Add methods to OOA classes

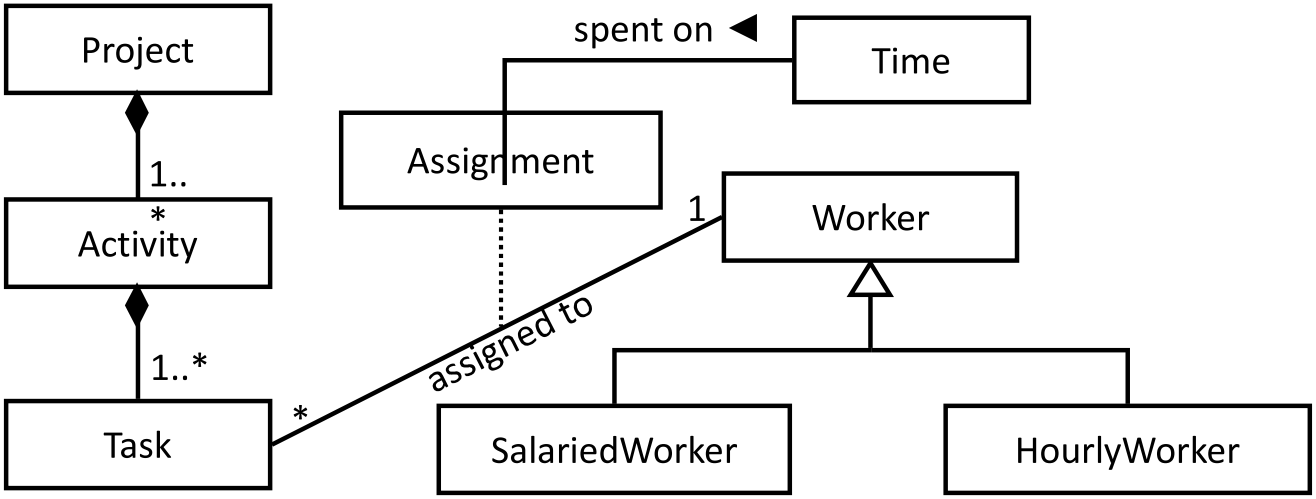
Example

- We are asked to build a system for keeping track of the time our workers spend working on customer projects.


Customer
Worker
*
assigned to
1
contracts
1
spends
*
*
*
Project
against
*
Time
*

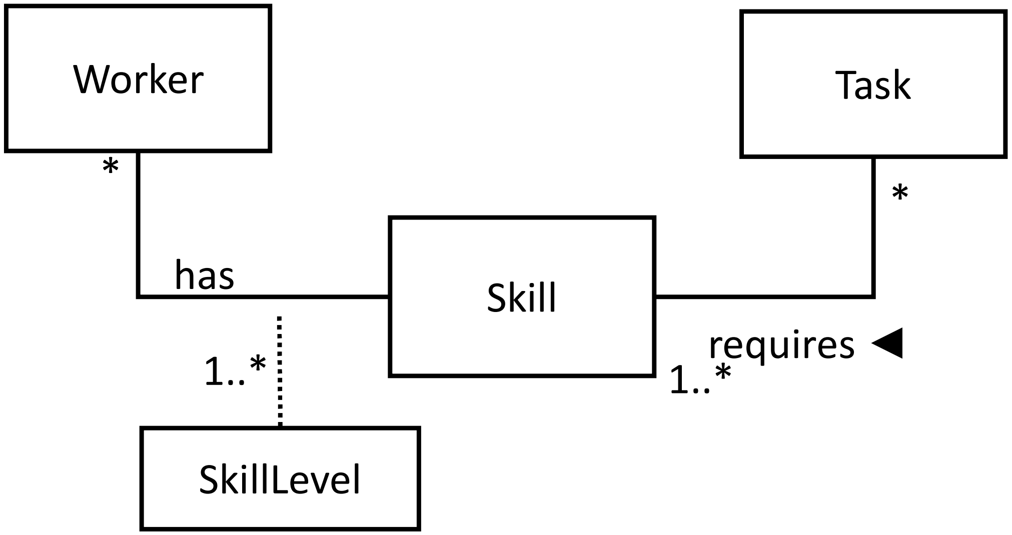
Example

- We divide projects into activities, and the activities into tasks.
- A task is assigned to a worker, who could be a salaried worker or an hourly worker.


Project
1..*
Activity
1..*
Task
*
Assignment
spent on ◀
Time
assigned to
1
Worker
SalariedWorker
HourlyWorker

Example

- Each task requires a certain skill, and workers have various skills at various level of expertise.


Worker
Task
*
*
has
Skill
requires ◀
1..*
1..*
SkillLevel

Steps

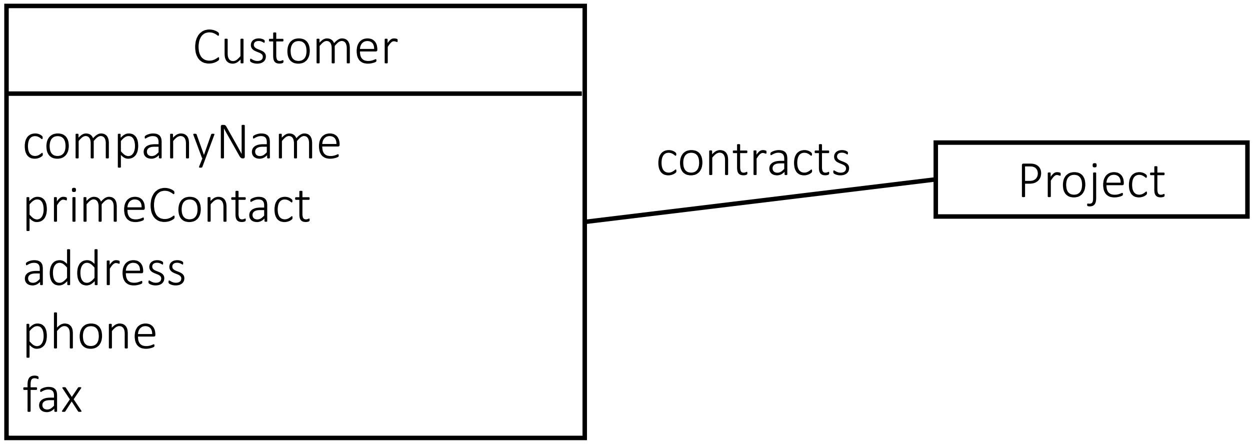
- Analyze the written requirements
  - Extract nouns: make them classes
  - Extract verbs: make them associations
  - Draw the OOA UML class diagrams

  - Determine attributes

  - Draw object diagrams to clarify class diagrams

- Determine the system's use cases
  - Identify Actors
  - Identify use case
  - Relate use cases

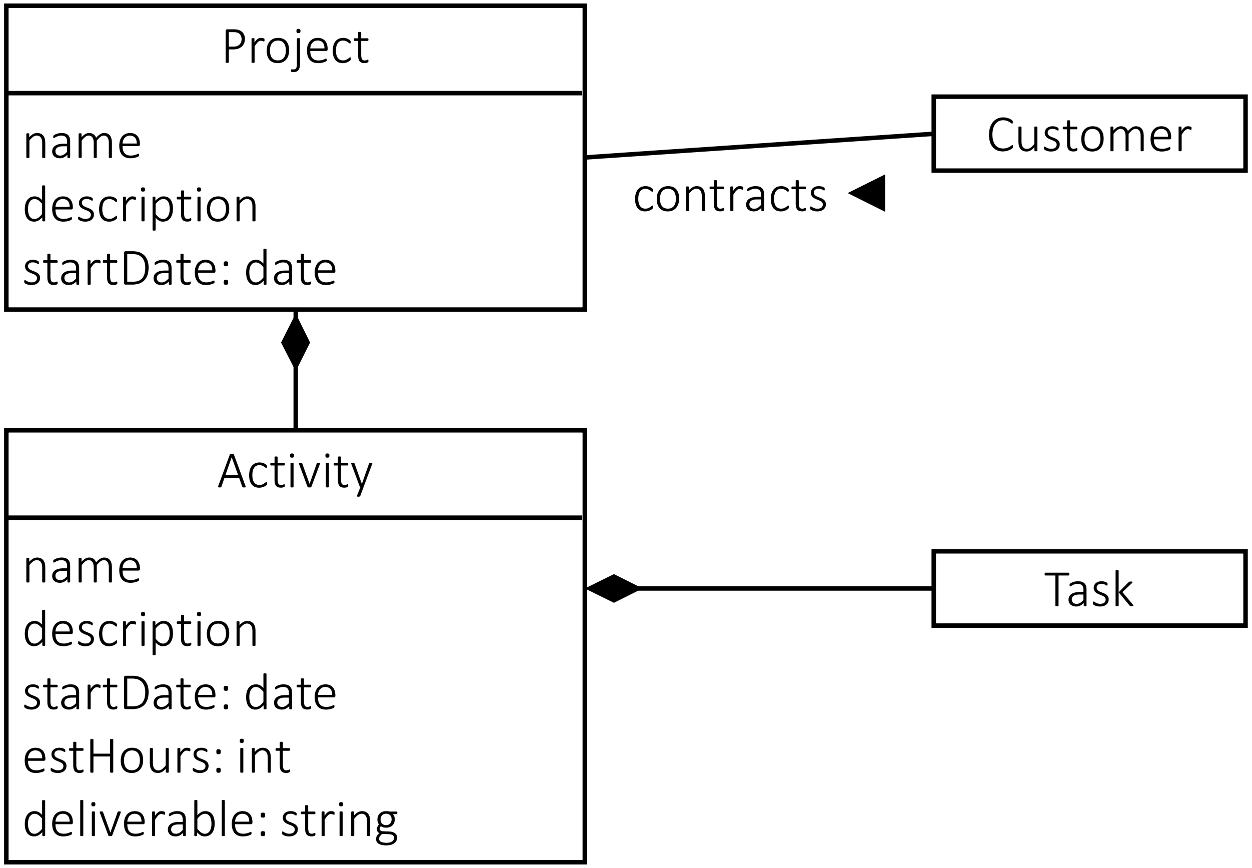
Example


Customer
companyName
primeContact
address
phone
fax
contracts
Project

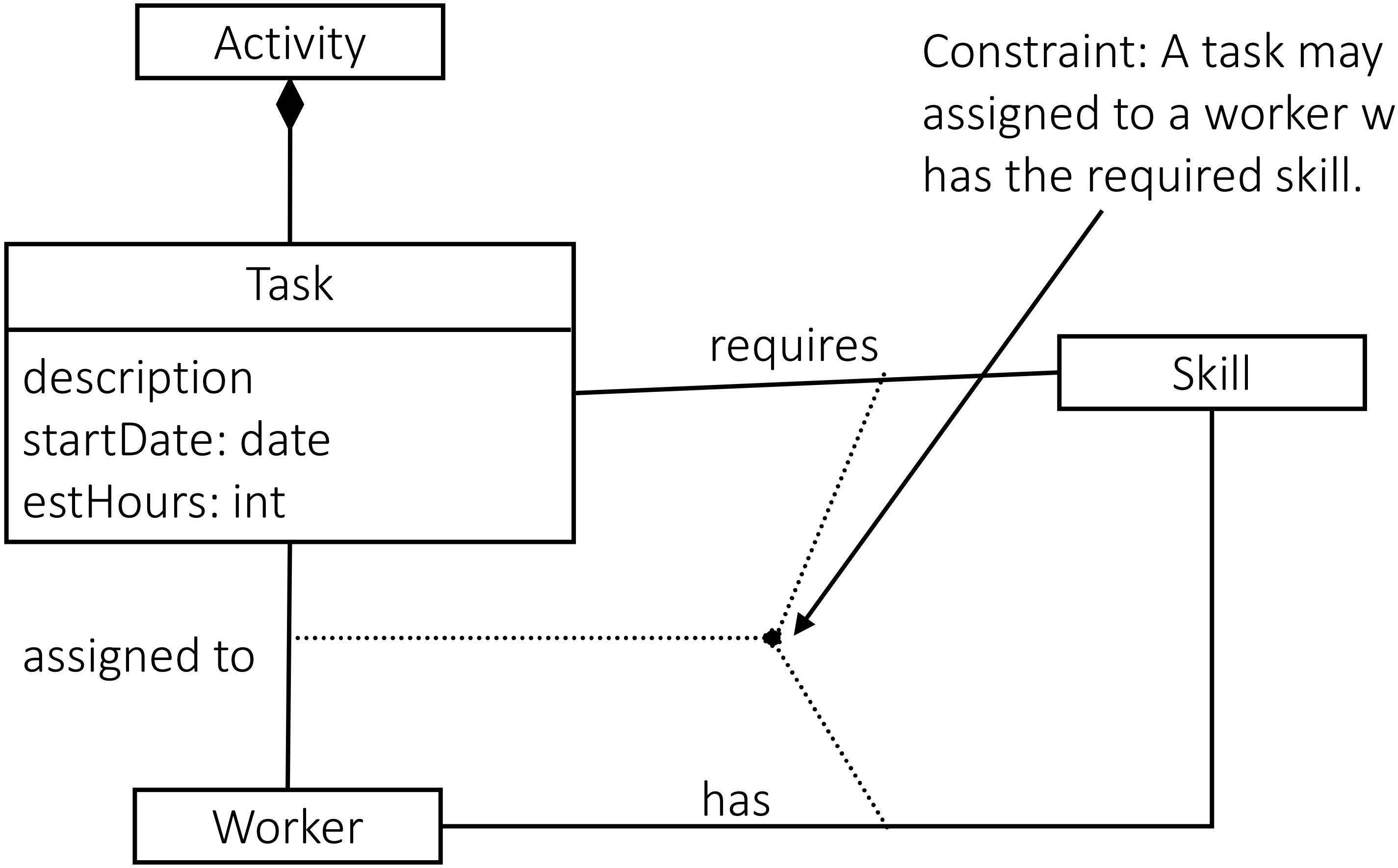
N.B.

- Project has no attribute in Customer
  - association is enough
- no database id for Customer shown
  - in an OOA, only include an id if visible to users
  - may include such things during database design or OOD

Example

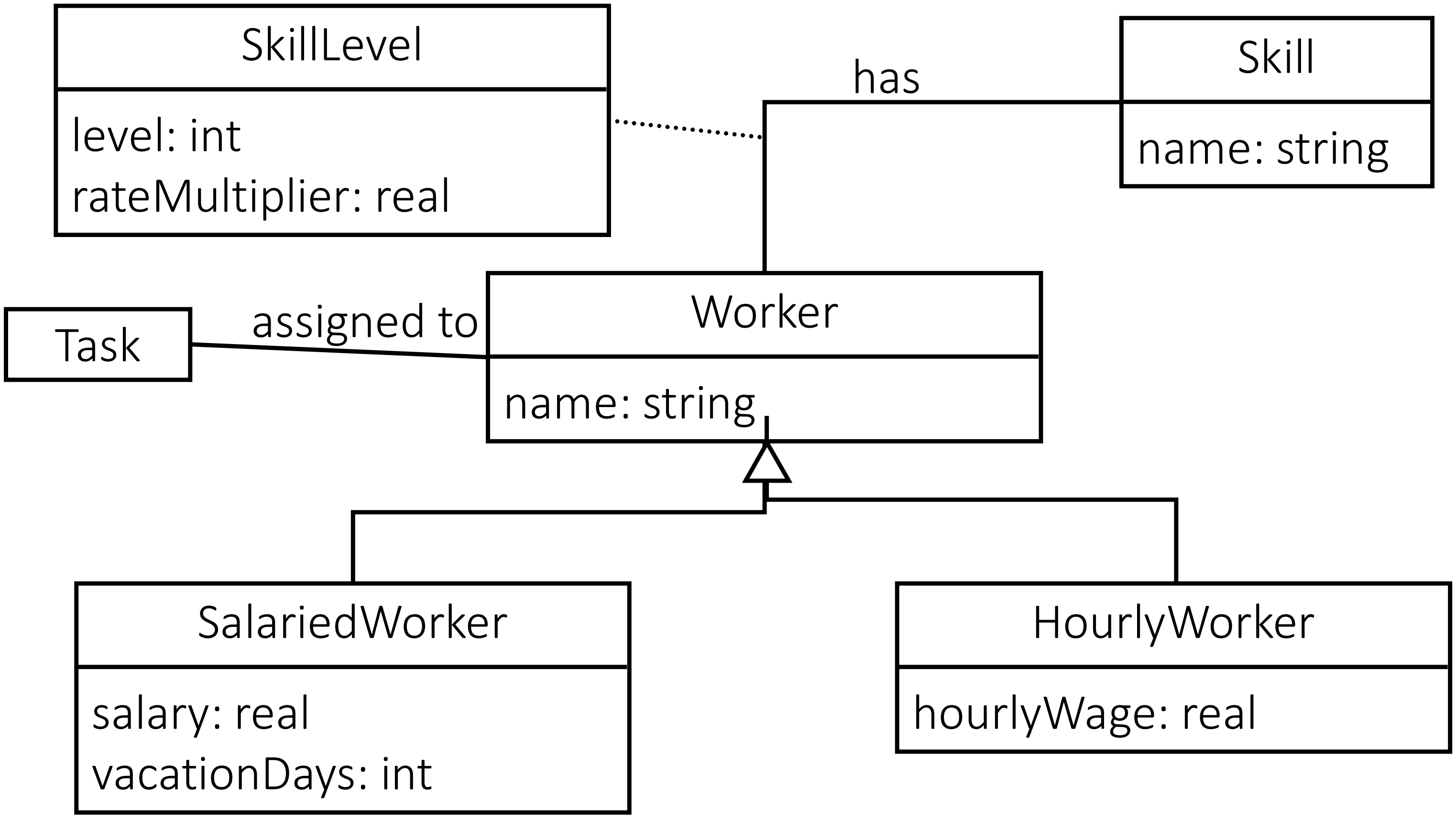

Project
name
description
startDate: date
Customer
contracts ◀
Activity
name
description
startDate: date
estHours: int
deliverable: string
Task

Example

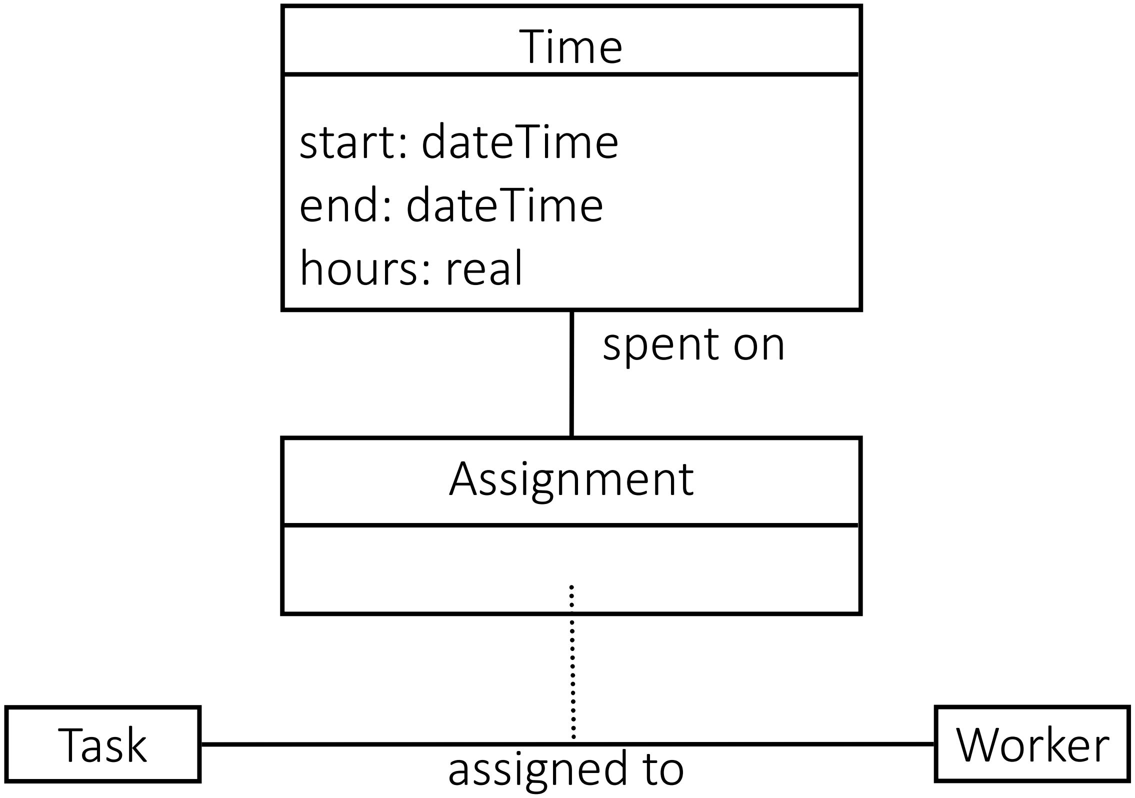

Activity
Task
description
startDate: date
estHours: int
requires
Skill
assigned to
has
Worker

Constraint: A task may only be
assigned to a worker who
has the required skill.

Example


SkillLevel
level: int
rateMultiplier: real
has
Skill
name: string
Task
assigned to
Worker
name: string
SalariedWorker
salary: real
vacationDays: int
HourlyWorker
hourlyWage: real

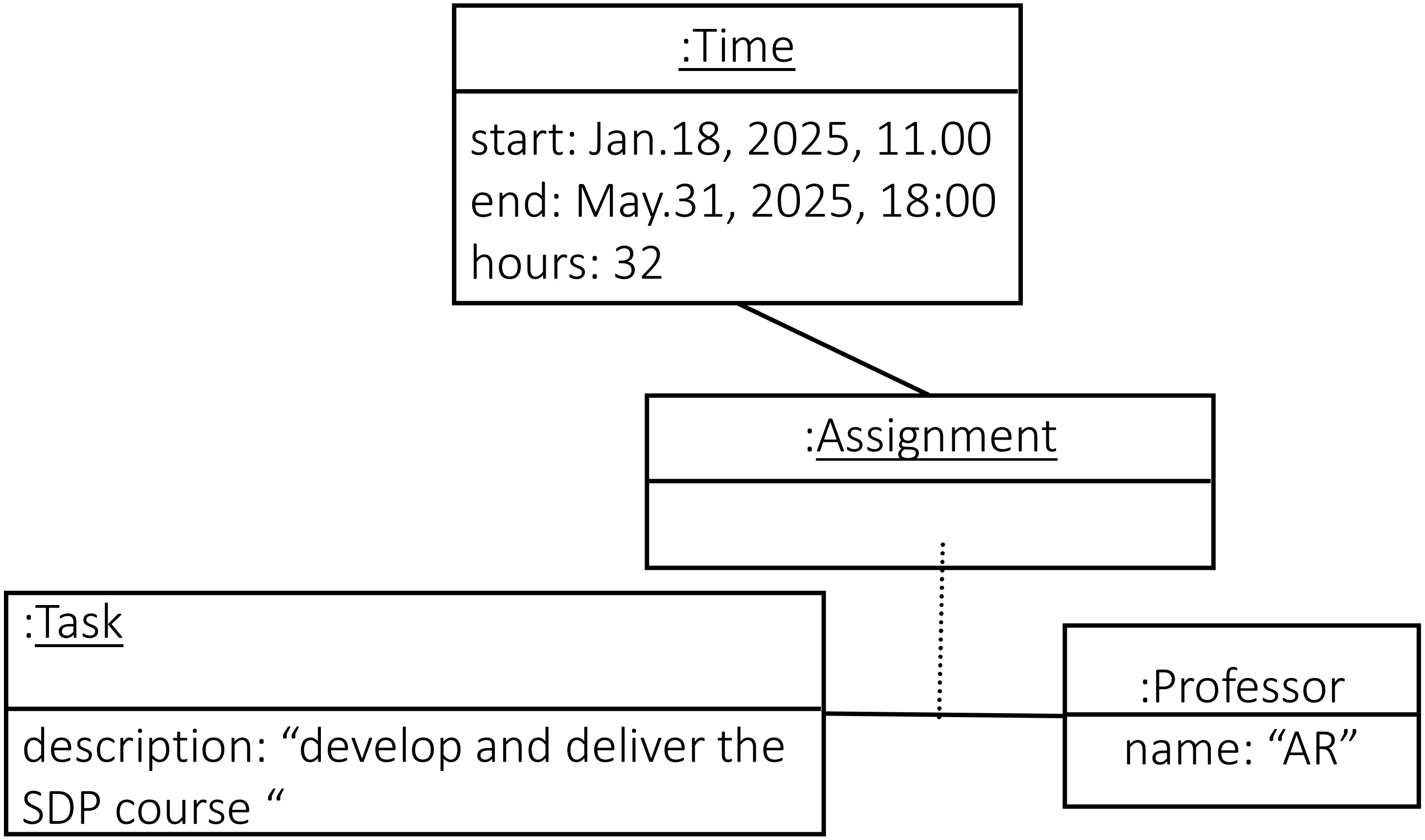
Example


Time
start: dateTime
end: dateTime
hours: real
spent on
Assignment
Task
assigned to
Worker

Steps

- Analyze the written requirements
  - Extract nouns: make them classes
  - Extract verbs: make them associations
  - Draw the OOA UML class diagrams
  - Determine attributes
  - Draw object diagrams to clarify class diagrams
- Determine the system's use cases
  - Identify Actors
  - Identify use case
  - Relate use cases
- Draw sequence diagrams
  - One per use case
  - Use to assign responsibilities to classes
- Add methods to OOA classes

Object Diagrams



:Time

start: Jan.18, 2025, 11.00
end: May.31, 2025, 18:00
hours: 32

:Assignment

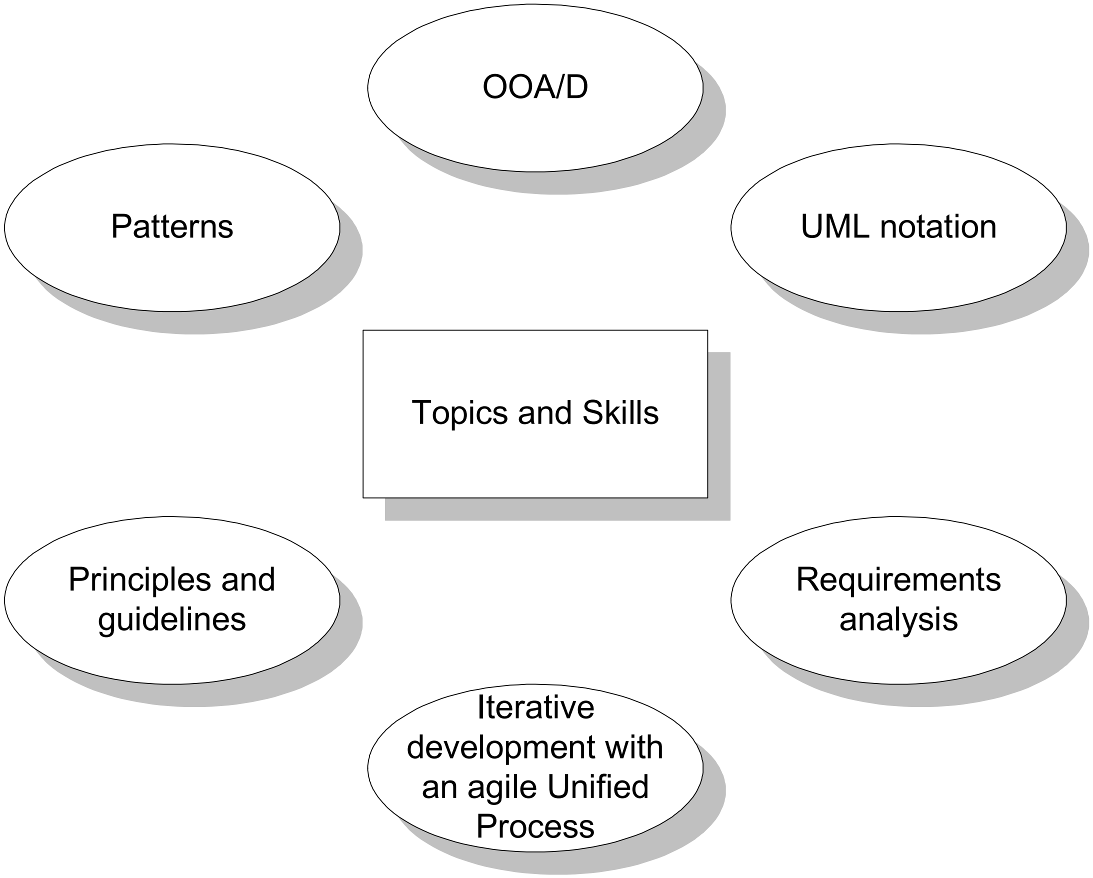
:Task

description: “develop and deliver the SDP course “

:Professor

name: “AR”

We will be studying all of the topics



OOA/D

Patterns

UML notation

Topics and Skills

Principles and guidelines

Requirements analysis

Iterative development with an agile Unified Process

The Rush to Code

- Critical ability to develop is to think in terms of objects and to artfully assign responsibilities to software objects.
- Talk at great length in about encapsulation and assigning methods to objects where the data is defined…
- One cannot design a solution if the requirements are not understood.
- One cannot implement the design if the design is faulty.

The Rush to Code

- Analysis: - investigate the problem and the requirements.
  - What is needed?
  - Required functions?
  - Investigate domain objects.
  - Problem Domain
  - The Whats of a system.
  - Do the right thing (analysis)

The Rush to Code

- Design:
  - Conceptual solution that meets requirements.
  - Not an implementation
  - E.g. Describe a database schema and software objects.
  - Avoid the CRUD activities and commonly understood functionality.
  - The Solution Domain
  - The ‘Hows’ of the system
  - Do the thing right (design)

What is Object-Oriented Analysis and Design

- OOA: we find and describe business objects or concepts in the problem domain

- OOD: we define how these software objects collaborate to meet the requirements.
  - Attributes and methods.

- OOP: Implementation: we implement the design objects in, say, Java, C++, C#, Python, etc.

Basic Terms: Iterative, Evolutionary, and Agile

- 1. Introduction
- *Iterative* - the entire project will be composed of min-projects and will iterate the same activities again and again (but on different part of the project AND with different emphases) until completion.

- *Evolutionary (*or *incremental)* - the software grows by increments (to be opposed to the traditional, and somewhat old-fashioned, Waterfall model of software development).

- *Agile* - we will use a light approach to software development rather than a very rigid one (which may be needed for a safety-critical systems)

- This kind of approach seems better at treating software development as a problem solving activity; also the use of objects makes it amenable.

Our Approach:

- We need a Requirements Analysis approach with OOA/OOD need to be practiced in a framework of a development process.

- We will adopt an agile approach (light weight, flexible) in the context of the Unified Process, which can be used as a sample iterative development process.
  - Within this process, the principles can be discussed.

- Please note that there are several other contexts that may be used, such as Scrum, XP, Feature-Driven Development, Lean Development, Crystal Methods and others…and we had looked at a few of these.

Why the Unified Process:

- The Unified Process is a popular iterative software development process.
- Iterative and evolutionary development involves relatively early programming and testing of a partial system, in repeated cycles.
- It typically also means that development starts before the exact software requirements have been specified in detail;
- Feedback (based on measurement) is used to clarify, correct and improve the evolving specification:
- This is in complete contrast to what we usually mean by engineering!

2. What is the Unified Process?

- The UP is very flexible and open and can include other practices from other methods such as Extreme Programming (XP) or Scrum for example.

  - e.g. XP's test-driven development, refactoring can fit within a UP project;
    So can Scrum's daily meeting.

  - Being pragmatic in adapting a particular process to your needs is an important skill : all projects are different.

The Unified Process – UP :

- A generic process framework that can be specialized
- Component-based
- Uses the Unified Modeling Language (UML).
- Use-case driven
- Architecture-centric
- Iterative and incremental

The Unified Process is use case driven

- Use cases capture functional requirements of the system.

- All the use cases together make up the use case model which describes the complete functionality of the system.

- Use cases derive the software development process!
  - Use-case driven

The Unified Process is Architecture centric

- Software architecture embodies the most significant static and dynamic aspects of the system.

- Software architecture is a view of the whole design with the important characteristics made more visible by leaving details aside.

Are There Other Critical UP Practices?

The central idea to appreciate and practice in the UP is short timeboxed iterative, evolutionary, and adaptive development. Some additional best practices and key concepts in the UP:

- tackle high-risk and high-value issues in early iterations
- continuously engage users for evaluation, feedback, and requirements
- build a cohesive, core architecture in early iterations
- continuously verify quality; test early, often, and realistically

Are There Other Critical UP Practices?

- apply use cases where appropriate
- do some visual modeling (with the UML)
- carefully manage requirements
- practice change request and configuration management

The Unified Process is Iterative and Incremental

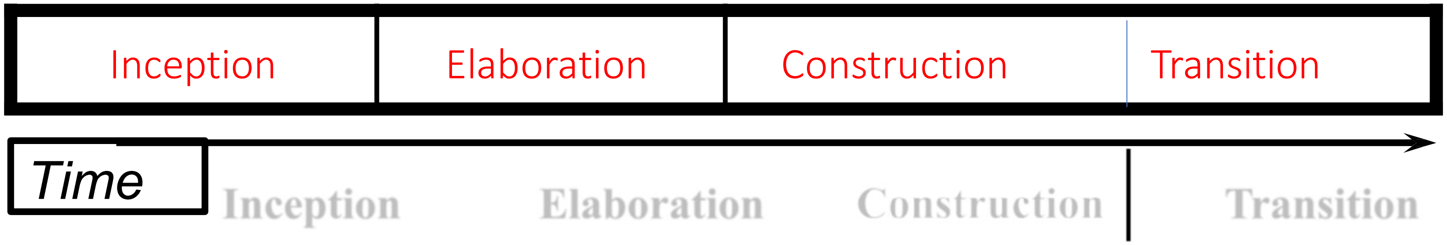
- It is practical to divide the software development work into smaller slices or mini-projects.

- Each mini-project is an iteration that results in an Increment.

- Iterations refer to steps in the workflow.

Four Phases in each cycle

- Each cycle consists of four phases:
    - Inception phase
    - Elaboration phase
    - Construction phase
    - Transition phase


Inception
Elaboration
Construction
Transition
Time
Inception
Elaboration
Construction
Transition

What are the UP Phases?

- A UP project organizes the work and iterations across four major phases:

1. Inception—approximate vision, business case, scope, vague estimates.

2. Elaboration—refined vision, iterative implementation of the core architecture, resolution of high risks, identification of most requirements and scope, more realistic estimates.

3. Construction—iterative implementation of the remaining lower risk and easier elements, and preparation for deployment.

4. Transition—beta tests, deployment.

What are the UP Phases?

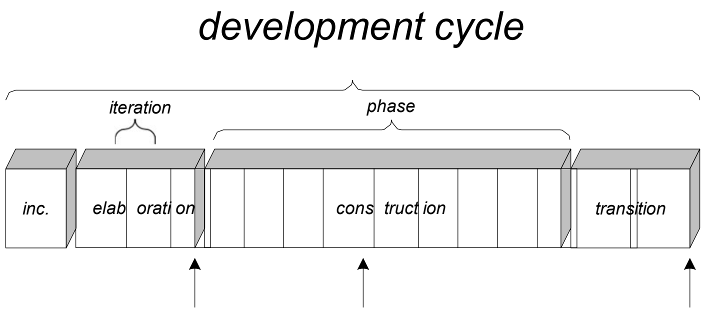
- This is *not* the old “waterfall” or sequential lifecycle of first defining all the requirements, and then doing all or most of the design.

- Inception is not a requirements phase; rather, it is a feasibility phase, where just enough investigation is done to support a decision to continue or stop.

- Similarly, elaboration is not the requirements or design phase; rather, it is a phase where the core architecture is iteratively implemented, and high-risk issues are mitigated.

one development cycle

illustrates common schedule-oriented terms in the UP


development cycle
iteration
phase
inc.
elaboration
construction
transition

milestone	release	increment	final production release
An iteration end- point when some significant decision or evaluation occurs.	A stable executable subset of the final product. The end of each iteration is a minor release.	The difference (delta) between the releases of 2 subsequent iterations.	At this point, the system is released for production use.

(which ends in the release of a system into production) is composed of many iterations.

What are the UP Disciplines?

- The UP describes work activities, such as writing a use case, within **disciplines**—

- a set of activities (and related artifacts) in one subject area, such as the activities within requirements analysis.

- In the UP, an **artifact** is the general term for any work product: code, Web graphics, database schema, text documents, diagrams, models, and so on.

- There are several disciplines in the UP; this book focuses on some artifacts in the following three:

■ **Business Modeling**—The Domain Model artifact, to visualize noteworthy concepts in the application domain.

What are the UP Disciplines?

- ■ Requirements—The Use-Case Model and Supplementary Specification artifacts to capture functional and non-functional requirements.

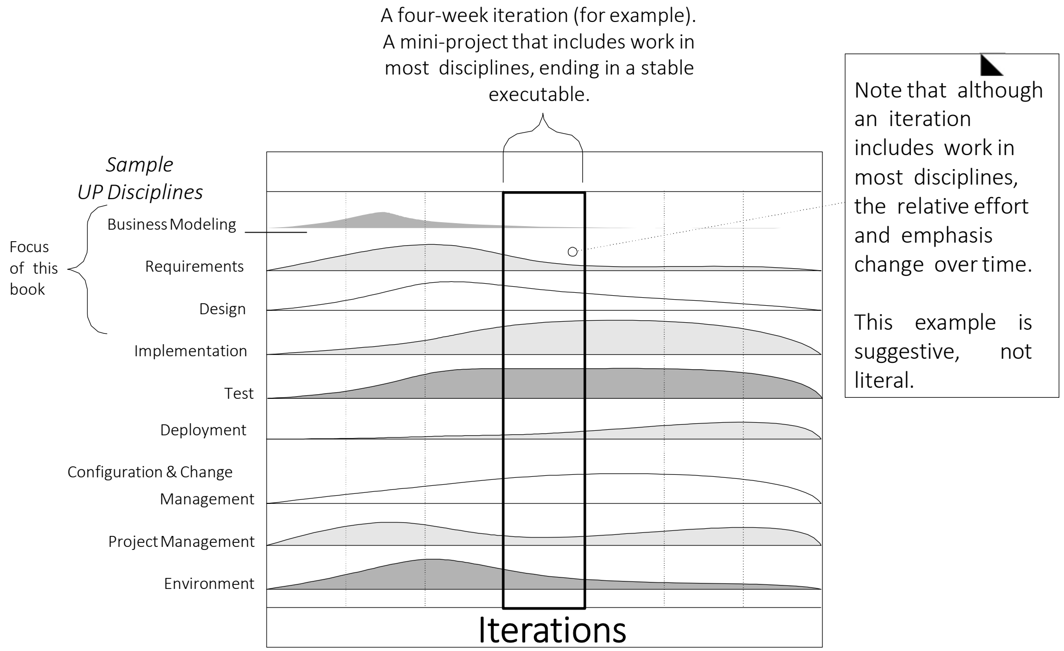
- ■ Design—The Design Model artifact, to design the software objects. A longer list of UP

disciplines is shown in Figure later

•In the UP, Implementation means programming and building the system, not deploying it.

•The Environment discipline refers to establishing the tools and customizing the process for the project—that is, setting up the tool and process environment.

What are the UP Disciplines?


A four-week iteration (for example).
A mini-project that includes work in most disciplines, ending in a stable executable.
Sample UP Disciplines
Focus of this book
Business Modeling
Requirements
Design
Implementation
Test
Deployment
Configuration & Change Management
Project Management
Environment
Iterations

Note that although an iteration includes work in most disciplines, the relative effort and emphasis change over time.

This example is suggestive, not literal.

What is the Relationship Between the Disciplines and Phases

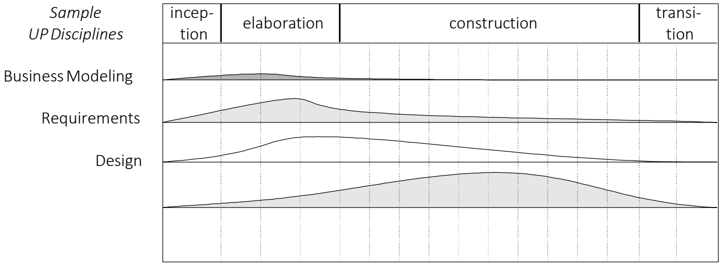
- As illustrated in the previous Figure, during one iteration work goes on in most or all disciplines. However, the relative effort across these disciplines changes over time.

- Early iterations naturally tend to apply greater relative emphasis to requirements and design, and later ones less so, as the requirements and core design stabilize through a process of feedback and adaptation.

- Relating this to the UP phases (inception, elaboration, …), illustrates the changing relative effort with respect to the phases; please note these are suggestive, not literal.

- In elaboration, for example, the iterations tend to have a relatively high level of requirements and design work, although definitely some implementation as well.
- During construction, the emphasis is heavier on implementation and lighter on requirements analysis.

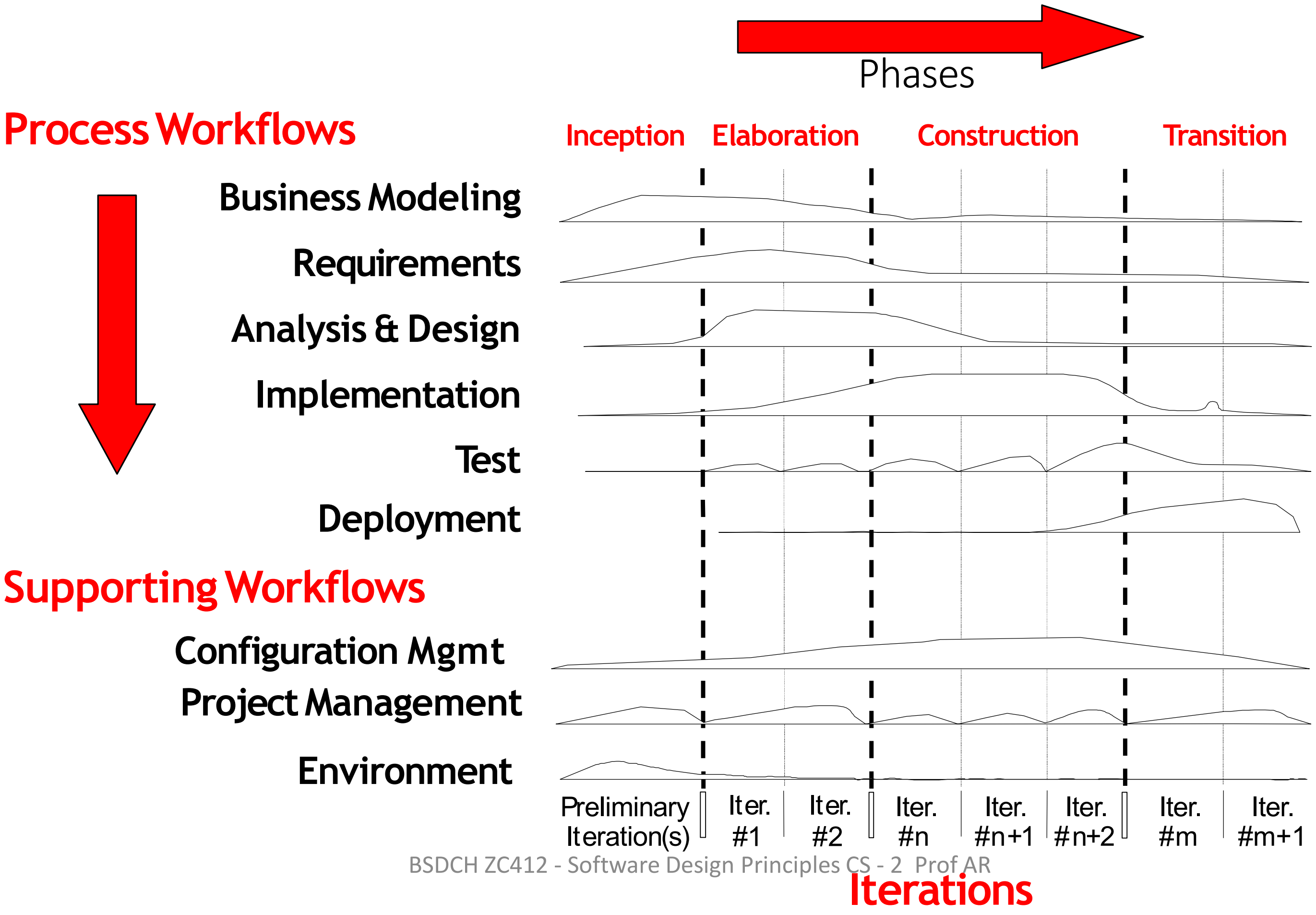
Disciplines and phases.



The relative effort in disciplines shifts across the phases.

This example is suggestive, not literal.

The structure of the Unified Process: Two Dimension

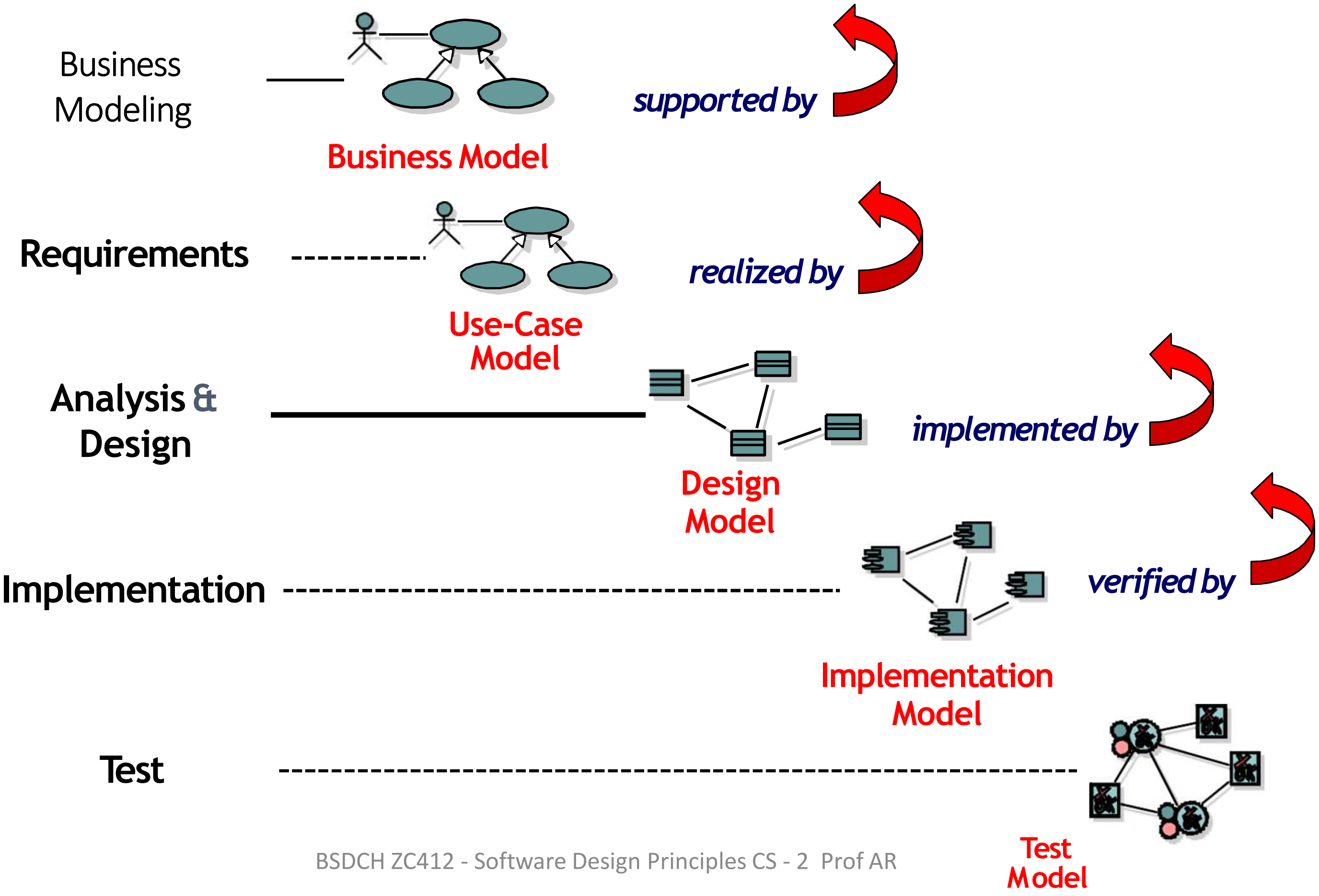

Phases
Process Workflows
Inception
Elaboration
Construction
Transition
Business Modeling
Requirements
Analysis & Design
Implementation
Test
Deployment
Supporting Workflows
Configuration Mgmt
Project Management
Environment
Preliminary Iteration(s)
Iter. #1
Iter. #2
Iter. #n
Iter. #n+1
Iter. #n+2
Iter. #m
Iter. #m+1
Iterations

The Iterations in each Phase

- Each phase is subdivided into iterations.

- An iteration is a distinct sequence of activities with an established plan and evaluation criteria, resulting in an executable release (internal or external).

The work flows produce Models



Business Modeling

Business Model

supported by

Requirements

Use-Case Model

realized by

Analysis & Design

Design Model

implemented by

Implementation

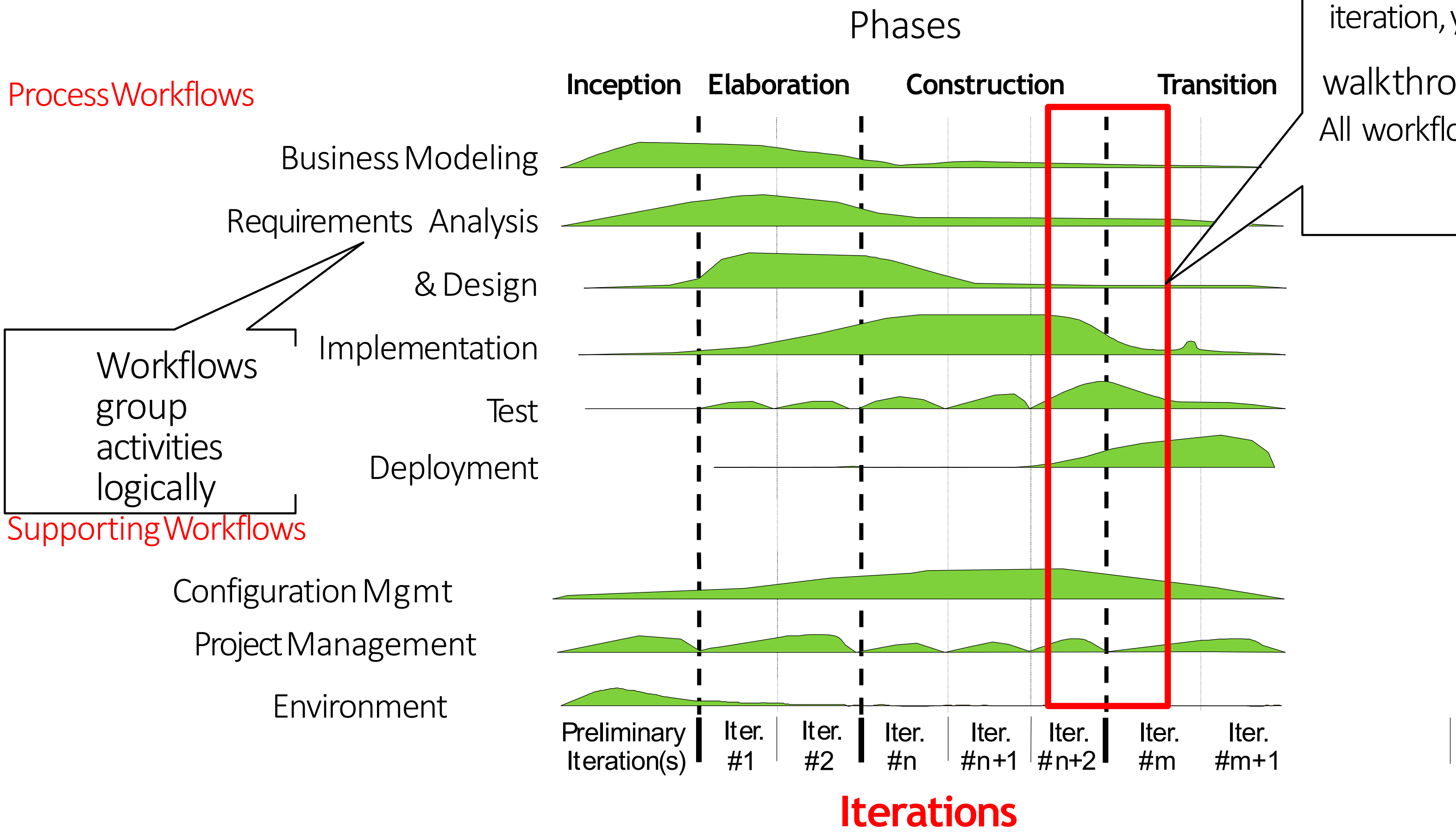
Implementation Model

verified by

Test

Test Model

Workflows, Phases and Iterations in the Unified Process



Phases

Inception **Elaboration** **Construction** **Transition**

Process Workflows

Business Modeling

Requirements Analysis

& Design

Implementation

Test

Deployment

Supporting Workflows

Configuration Mgmt

Project Management

Environment

Preliminary Iteration(s) | Iter. #1 | Iter. #2 | Iter. #n | Iter. #n+1 | Iter. #n+2 | Iter. #m | Iter. #m+1

Iterations

Workflows group activities logically

In an iteration, you walkthrough All workflows

The Unified Process - Summary

- Unified Process: Iterative software lifecycle model
    - Emphasis on early construction of a software architecture
    - Emphasis on early demonstrations of the system

    - Phase: Status of the software system.
- 4 phases: Inception, Elaboration, Construction, Transition

- Workflow: Mostly sequential activity that produces artifacts
        - 7 workflows: Management, environment, requirements, design, implementation, assessment, deployment.
        - 5 artifact sets: Management set, requirements set, design set, implementation set, deployment set
- Iteration: Repetition within a workflow.

BITS Pilani
Pilani Campus

SOFTWARE DEVELOPMENT LIFECYCLE

Use cases An Overview

By A R Rahman

USE CASE DIAGRAM

INTRODUCTION

- Use cases – Actors, goals (needs), scenarios
- Different formats to write use cases

USE CASE Diagram

- A use case diagram establish the capability of the system as a whole.

- Graphical specification of the system’s behavior from the perspective of the user(s).

- Describes what the system does without describing how the system does it.

- They are used to identify and communicate the high-level business requirements for the system.

System Flow

Requirements phase

- Gather requirements
  - Description / Screen shots
- Use – case interactions
- Textual description of use cases

Analysis phase

Identify the structure from the use case

Collaboration diagrams from the use case

Design phase

- Class diagrams
- Adapt to actual environment

USE CASE Diagram

- Used for describing a set of user scenarios
- Mainly used for capturing user requirements
- Provide the basis for identifying major classes and their relationships
- Serve as the foundation for developing system test cases
- Functional requirements, Analysis Phase

Use Case Diagram

- Captures system functionality as seen by users
- Built in early stages of development Purpose
- Specify the context of a system
- Capture the requirements of a system

Use Case Diagram

- Validate a system's architecture
- Drive implementation and generate test cases
- Developed by analysts and domain experts

What is USE CASE ?

- A view of a system that emphasizes the behavior as it appears to outside users.

- A use case model partitions system functionality into transactions (i.e., use cases) that are meaningful to users (i.e., actors).

- A use case is a scenario that describes the use of a system by an actor to accomplish a specific goal

What is USE CASE ? …

- System behavior is how a system acts and reacts.
  - It is an outwardly visible and testable activity of a system.
- System behavior is captured in use cases.
  - Use cases describe the system, its environment, and the relationship between the system and its environment.

Components of USE CASE diagram

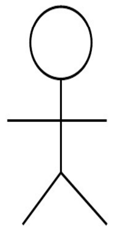
- Actor
- Use case
- System boundary
- Relationship
- Actor relationship

Actors

Roles played by the users of the system can be humans, other computers, pieces of hardware, or even other software systems.

External to the parts of the system

Supply input to the system and receive output from the system



Scenario ..

- A scenario is a sequence of steps describing an interaction between a user and a system.

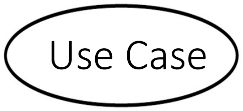
- Specify behaviour of use case by description, not modeling

- Examples include informal structured text, formal structured text with conditions, and pseudocode

A typical use of a System by an actor involving one or more sequences of actions in the system on behalf of the user

Capture of a system functionality in a use case

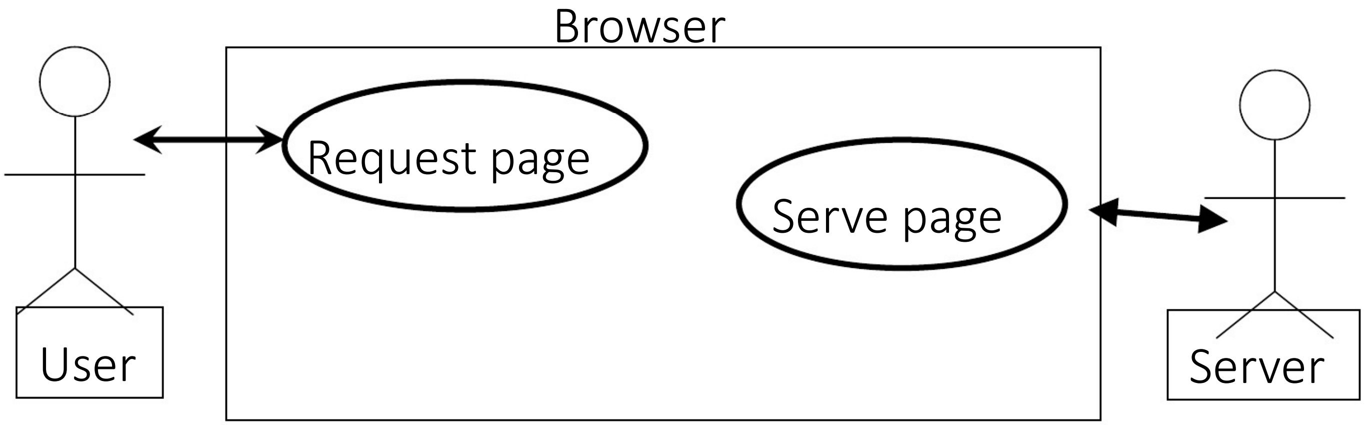
Use case realized by a collaboration in the system



Use case instance manifests as a scenario

A use case encompasses all possible scenarios for it

Use case Example



Browser

Request page

Serve page

User

Server

Scenario ..

Typically specifies:

- How and when the use case starts and ends
- Interaction with the actors and the exchange of objects
- Flow of events: main / typical (success),

  alternative (success), and exceptional (failure) flows

- However, the credit card authorization might fail. This would be a separate scenario.

Scenarios Example

- In a human resources system, for the “Hire Employee” use case, the following scenarios may apply:

- **Typical success scenario**: Hire a person from outside of the company, for example, from another company

- **Alternative success scenario**: Hire a person from within the same company, for instance, from another division

- **Exceptional failure scenario:** No qualified person could be hired

Scenarios Example

- For example, if we have a Web-based on-line store, we might have a Buy a Product scenario that would say this:
- *The customer browses the catalog and adds desired items to the shopping basket.*
- *When the customer wishes to pay, the customer describes the shipping and credit card information and confirms the sale.*
- *The system checks the authorization on the credit card and confirms the sale both immediately and with a follow-up email.*

Conditions and Quality Requirements

- Entry and Exit Conditions
  - Entry conditions describe the environment under which the use case is invoked
  - Exit conditions reflect the impact of the use case on the environment through its execution
- Quality Requirements
  - Describe quality attributes in terms of a specific functionality
  - For example, requires system response in < 30 seconds

Use Case Modeling: Core Elements

Construct	Description	Syntax
use case	A sequence of actions, including variants, that a system (or other entity) can perform, interacting with actors of the system.	
actor	A coherent set of roles that users of use cases play when interacting with these use cases.	
system boundary	Represents the boundary between the physical system and the actors who interact with the physical system.	

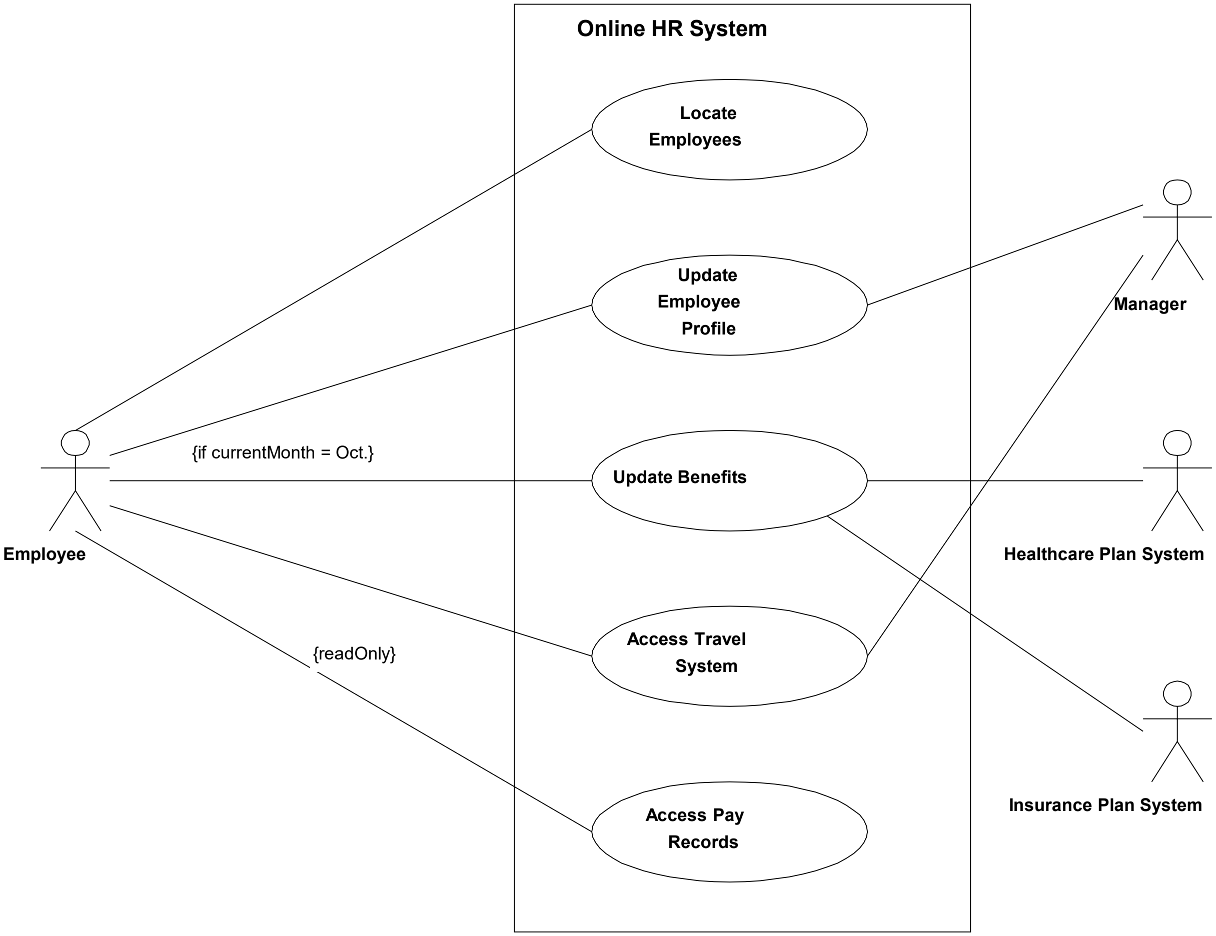
Use Case Modeling: Core Relationships

Construct	**Description**	**Syntax**
association	The participation of an actor in a use case. i.e., instance of an actor and instances of a use case communicate with each other.	
extend	A relationship from an *extension* use case to a *base* use case, specifying how the behavior for the extension use case can be inserted into the behavior defined for the base use case.	
generalization	A taxonomic relationship between a more general use case and a more specific use case.	

Use Case Modeling: Core Relationships

Construct	Description	Syntax
include **Or** **use**	An relationship from a *base* use case to an *inclusion* use case, specifying how the behavior for the inclusion use case is inserted into the behavior defined for the base use case.	

Example: Online HR System


Online HR System
Locate Employees
Update Employee Profile
Update Benefits
Access Travel System
Access Pay Records
Employee
Manager
Healthcare Plan System
Insurance Plan System
{if currentMonth = Oct.}
{readOnly}

Actors

- An actor is some one or something that must interact with the system under development

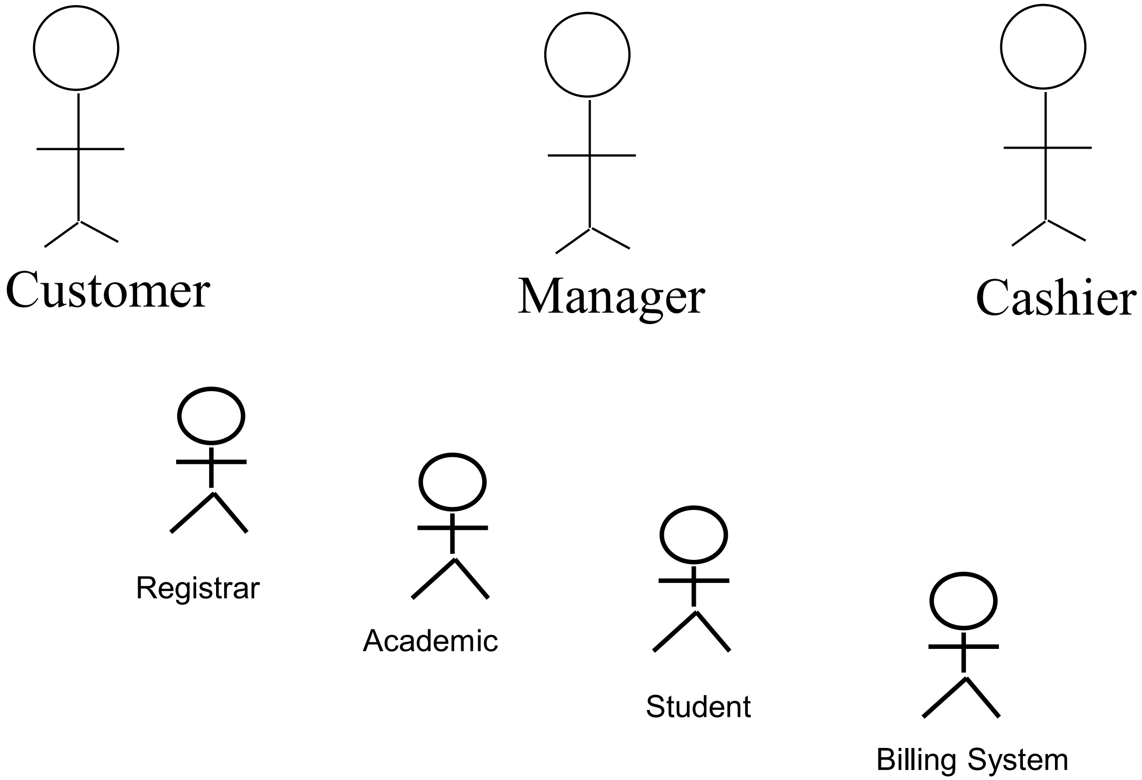
- A role that a user plays with respect to the system ,including human users and other systems.

- e.g., inanimate physical objects (e.g. robot)

  an external system that needs some information from the current system

Actors

- UML notation for actor is stickman, shown below.


Customer
Manager
Cashier
Registrar
Academic
Student
Billing System

More on Actors …

- It is role a user plays with respect to system.

- Actors are not part of the system they represent anyone or anything that must interact with the system.

- Actors carry out use cases and a single actor may perform more than one use cases.

- Actors are determined by observing the direct uses of the system,

More on Actors …

- Those are responsible for its use and maintain as well as other systems that interact with the developed system

- An actor may

  - input information to the system.

  - receive information from the system.

  - input to and out from the system.

Find Actors

- Ask following questions to find the actors:
  - Who uses the system?
  - Who installs the system?
  - Who Starts up the system?
  - What other systems use this system?
  - Who gets the information from the system?
  - Who provides information to the system?
- Actor is always external to the system. They are never part of the system to be developed.

Categories of an actor

There are 4-Categories of an actor:

- Principle : Who uses the main system functions.
- Secondary : Who takes care of administration & maintenance.
- External h/w : The h/w devices which are part of application domain and must be used.
- Other system : The other system with which the system must interact.

Note on Actors

- If newly identified actor is using a system in a same way like an existing actor, then new actor can be dropped.

- If two actors use system in the same way they are same actors

USE CASE

- A use case is a pattern of behavior, the system exhibits

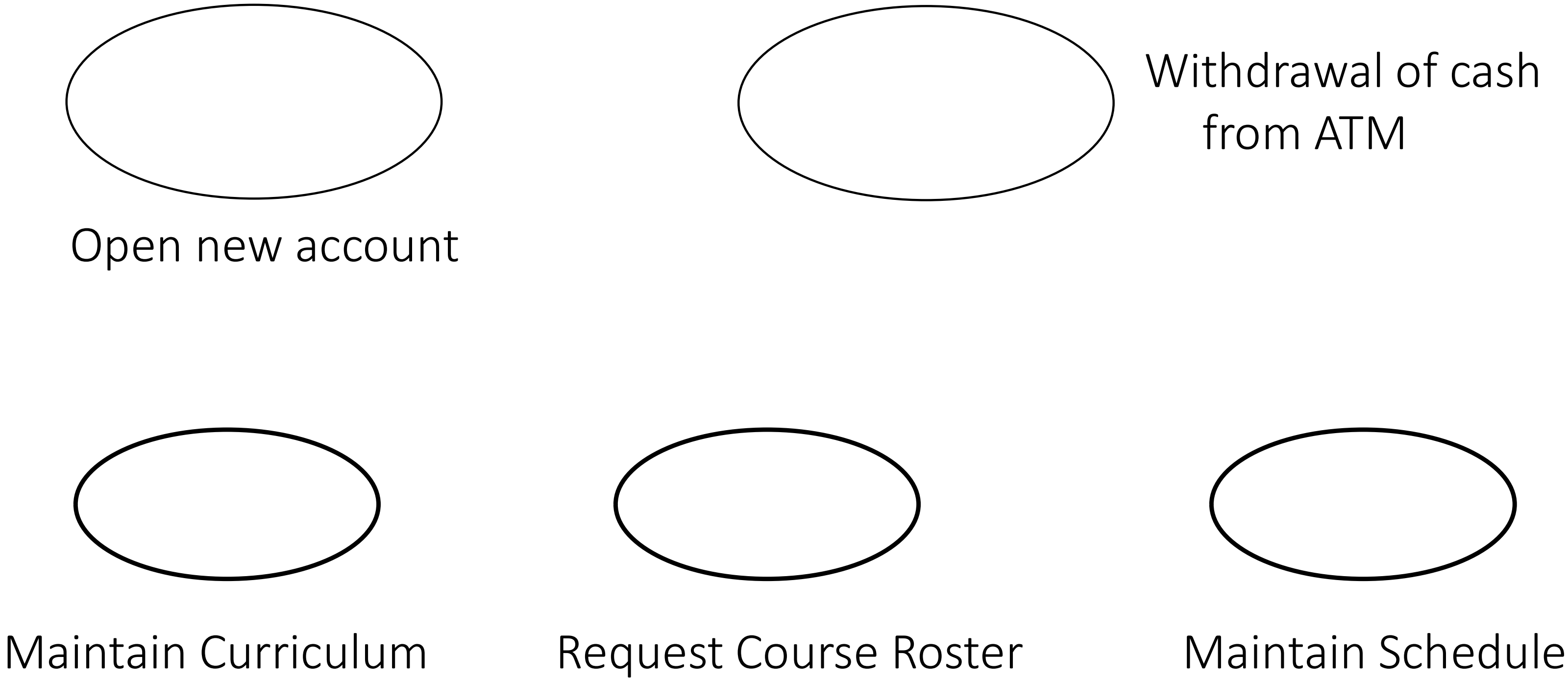
- Each use case is a sequence of related transactions performed by an actor and the system in dialogue.

- USE CASE is dialogue between an actor and the system.

- A use case describes a sequence of actions that provide something of measurable value to an actor and is drawn as a horizontal ellipse.

USE CASE …

- Examples:



Withdrawal of cash
from ATM

Open new account

Maintain Curriculum

Request Course Roster

Maintain Schedule

USE CASE …

Actors are examined to determine their needs

- Registrar -- maintain the curriculum
- Professor -- request roster
- Student -- maintain schedule
- Billing System -- receive billing information from registration

USE CASE

- It is a snapshot of one aspect of system.

- They model a dialog between actor and system.

- A use case typically represents a major piece of functionality that is complete from beginning to end.

- Most of the use cases are generated in initial phase, but you find some more as you proceed..

USE CASE

- A use case may be small or large.
- It captures a broad view of a primary functionality of the system in a manner that can be easily grasped by non technical user
- A use case must deliver something of value to an actor.
- The use cases may be decomposed into other use cases.
- Use cases also present a good vehicle for project planning

FIND USE CASES..

- What functions will the actor want from the system?

- Does the system store information? What actors will create, read, update. Or delete that information?

- Does the system need to notify an actor about changes in its internal state?

USE CASE…

- Generic format for documenting the use case:
  - Pre condition: If any
  - Use case : Name of the case.
  - Actors : List of actors(external agents), indicating who initiates the use case.
  - Purpose : Intention of the use case.
  - Overview : Description.
  - Type : primary / secondary.
  - Post condition: If any
- Typical Course of Events:

ACTOR ACTION : Numbered actions of the actor.

SYSTEM RESPONSE : Numbered description of system responses.

USE CASE documentation example

The following use case describes the process of opening a new account in the bank.

Use case :Open new account
Actors : Customer, Cashier, Manager
Purpose : Like to have new saving account.
Description : A customer arrives in the bank to open the new account.
Customer requests for the new account form,
fill the same and submits,
along with the minimal deposit.
At the end of complete successful
process customer receives the passbook.
Type : Primary use case.

Detail an Use case

Use-case model and use-case diagrams used as starting point

Step-by-step description of each use case

Flow of Events

- How to Structure to specify all alternative paths?
- What to include in a description?
- How to formalize the description?

Structuring Use-Case Description

States that use-case instances may enter and possible transitions between states

Each such transitions is a sequence of actions

May get complicated and intricate

Precise and simple description needed
Start with a Basic Path
start to end state in one section of description

Alternate Paths described in separate section

If small, may inline in Basic Path

Goal: Precise description that is easy to read

What to include in description

Start state as precondition

How and when use-case starts (step 1)

Order for the flow of events

How and when use-case ends

Possible end state as post condition

Paths of execution that is not allowed In-lined alternative paths

Alternative path descriptions extracted from basic path

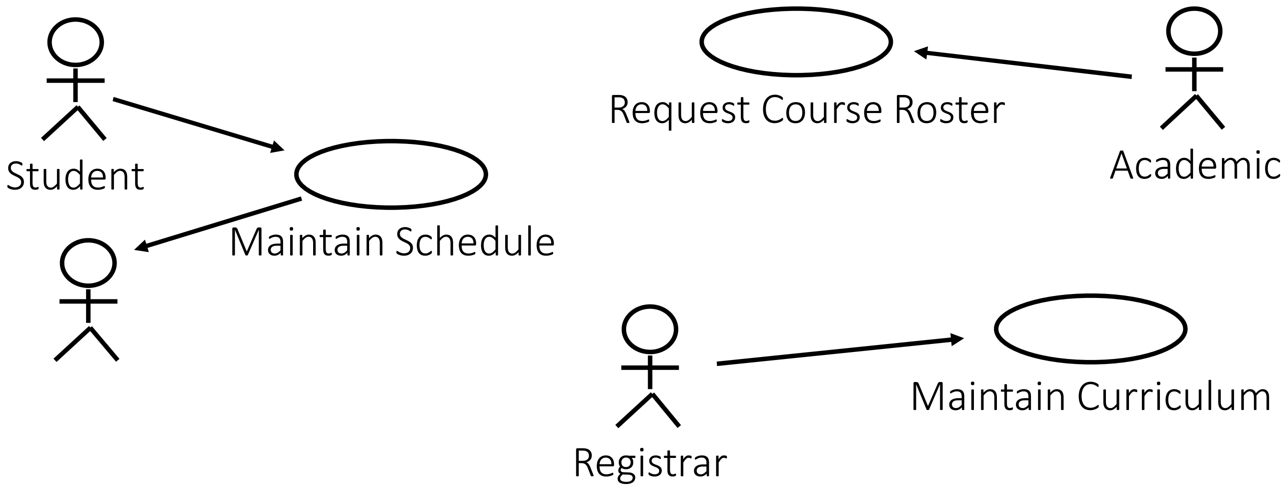
System interaction with actor

Usage of objects, values, resources in system

Explicitly describe what system does

USE CASE DIAGRAM EXAMPLE

- Use case diagrams are created to visualize the relationships between actors and use cases


Request Course Roster
Academic
Student
Maintain Schedule
Registrar
Maintain Curriculum

USE CASE DIAGRAM EXAMPLE

- In a banking application, a deposit function represents a use case.

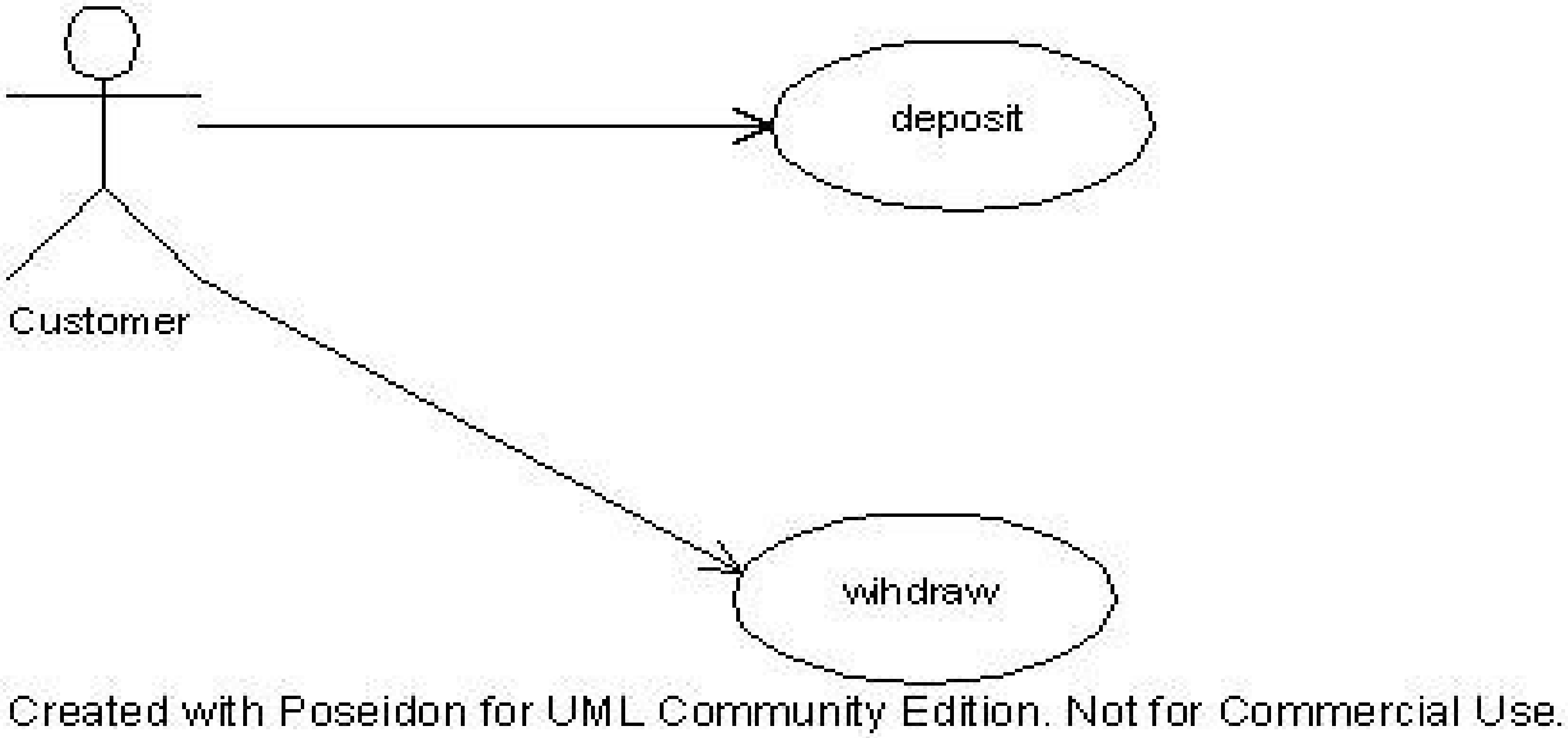
- To identify use cases, you should list the discrete business functions the user might do in order to complete an action.

- For a banking application it might be:

  deposit money

  withdraw money

USE CASE DIAGRAM EXAMPLE


Customer
deposit
wihdraw
Created with Poseidon for UML Community Edition. Not for Commercial Use.

SYSTEM BOUNDARY

- It is shown as a rectangle.

- It helps to identify what is external verses internal, and what the responsibilities of the system are.

- The external environment is represented only by actors

RELATIONSHIP

- Relationship between use case and actor.

  Communicates

- Relationship between two use cases

  Extends

  Include

- Notation used to show the relationships:

  << >>

RELATIONSHIP

- Relationship between use case and actor is often referred as “communicates” .

- Relationship between two use cases is refereed as either include or extends.

- Also using Use Case Generalization

- Both generalization and extend allow to split a use case.

INCLUDE ..

- Multiple use cases share a piece of same functionality.

- This functionality is placed in a separate use case rather than documenting in every use case that needs it.

- A INCLUDE relationship shows behavior that is common to one or more use cases.

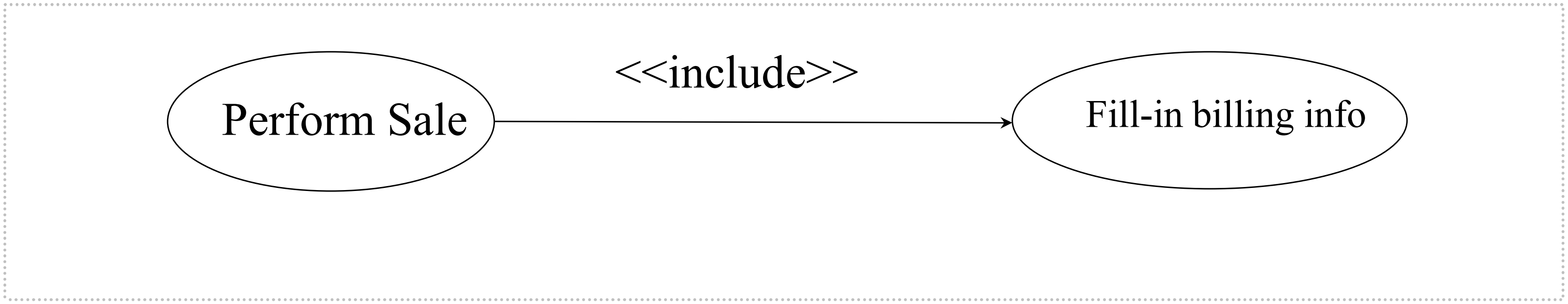
Include

- Include: a dotted line labeled <<include>> beginning at base use case and ending with an arrows pointing to the include use case.
- The include relationship occurs when a chunk of behavior is similar across more than one use case.
- Use "include" in stead of copying the description of that behavior.
- <<include>>

The “Include” Construct

- Include is used when:
  - Decomposing complicated behavior
  - Centralizing common behavior

- The base use case explicitly incorporates the behavior of another use case at a location specified in the base.



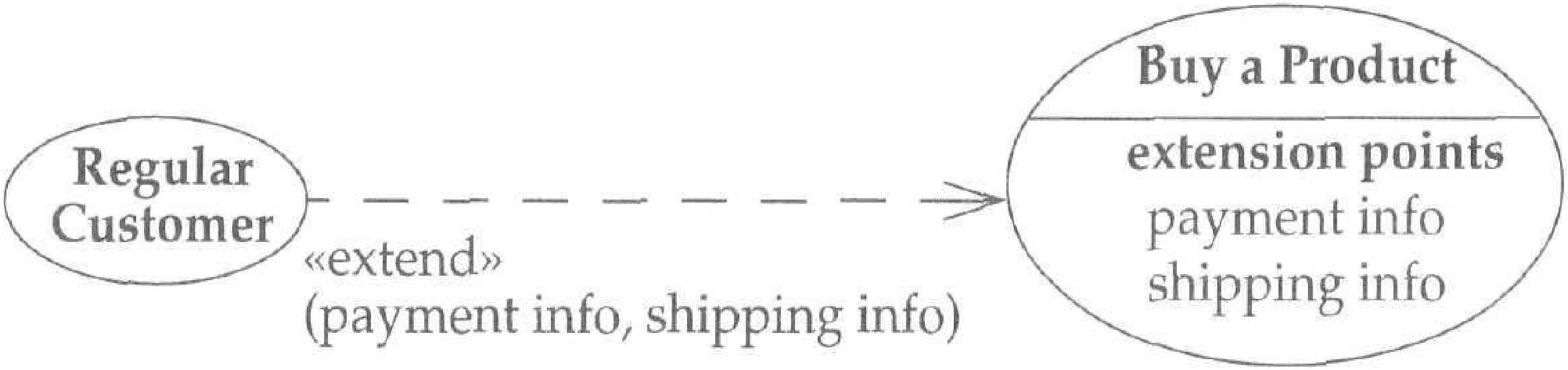
Perform Sale

<<include>>

Fill-in billing info

EXTENDS

- It is used to show optional behavior, which is required only under certain condition
- Extending use case may add behavior to the base use case
- i.e., the base use case must declare certain "extension points," and the extending use case may add additional behavior only at those extension points


Regular Customer
«extend»
(payment info, shipping info)
Buy a Product
extension points
payment info
shipping info

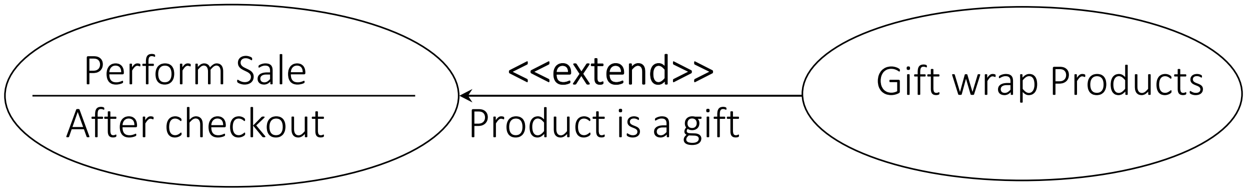
EXTENDS

- Extend: a dotted line labeled <<extend>> with an arrow toward the base case.
- The extending use case may add behavior to the base use case.
- The base class declares "extension points".
- <<extend>>

Extend – Graphical Representation

- The base use case can incorporate another use case at certain points, called extension points.

- Note the direction of the arrow
  - The base use-case does not know which use-case extends it


Perform Sale
After checkout
<<extend>>
Product is a gift
Gift wrap Products

Example

USE CASE GENERALIZATION

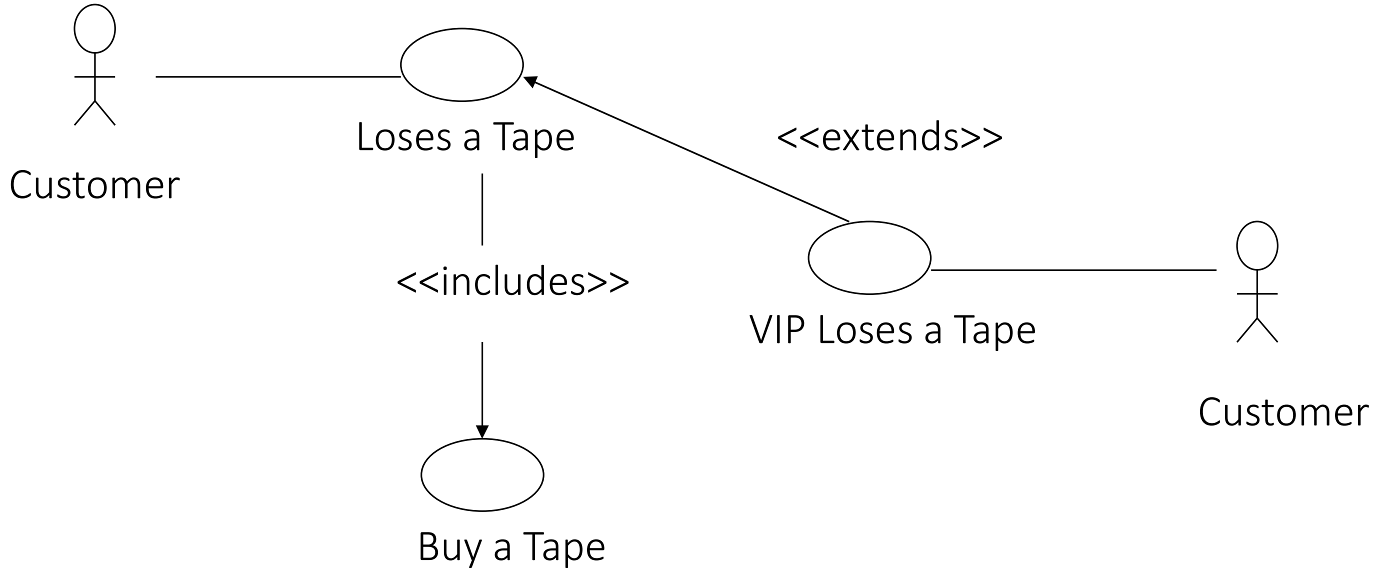
Use case generalization

- Third relationship
- When we have one use case that is similar to another use case but does a bit more.
- In effect, this gives us another way of capturing alternative scenarios.

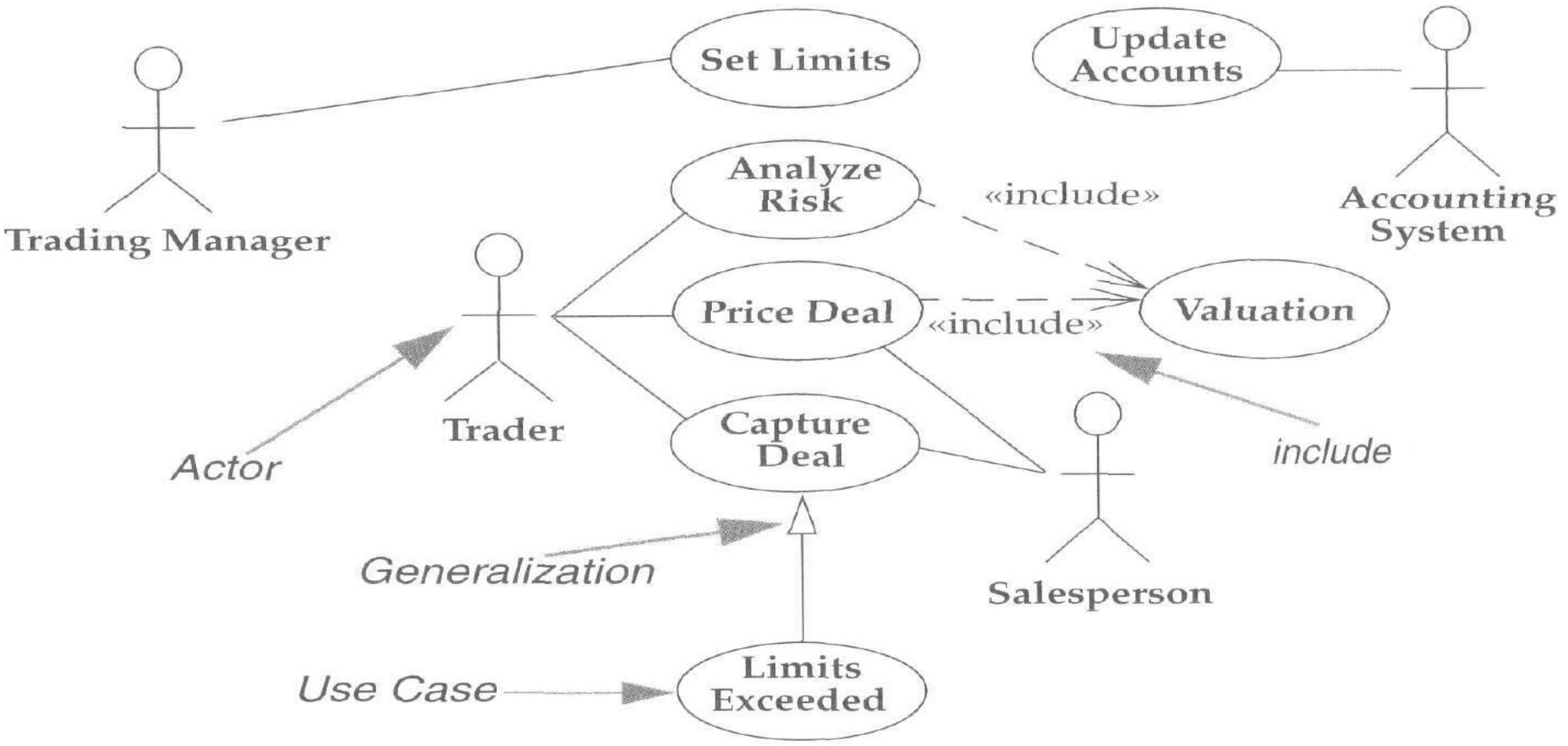
Use Case relationships

<<includes>>: one use case always involves the steps of another

<<extends>>: under certain conditions, a use case follows a variant


Customer
Loses a Tape
<<extends>>
<<includes>>
VIP Loses a Tape
Customer
Buy a Tape

USE CASE DIAGRAM EXAMPLE


Trading Manager
Set Limits
Update Accounts
Accounting System
Analyze Risk
«include»
Trader
Price Deal
«include»
Valuation
Capture Deal
Salesperson
Actor
include
Generalization
Use Case
Limits Exceeded

RELATIONSHIP RULES

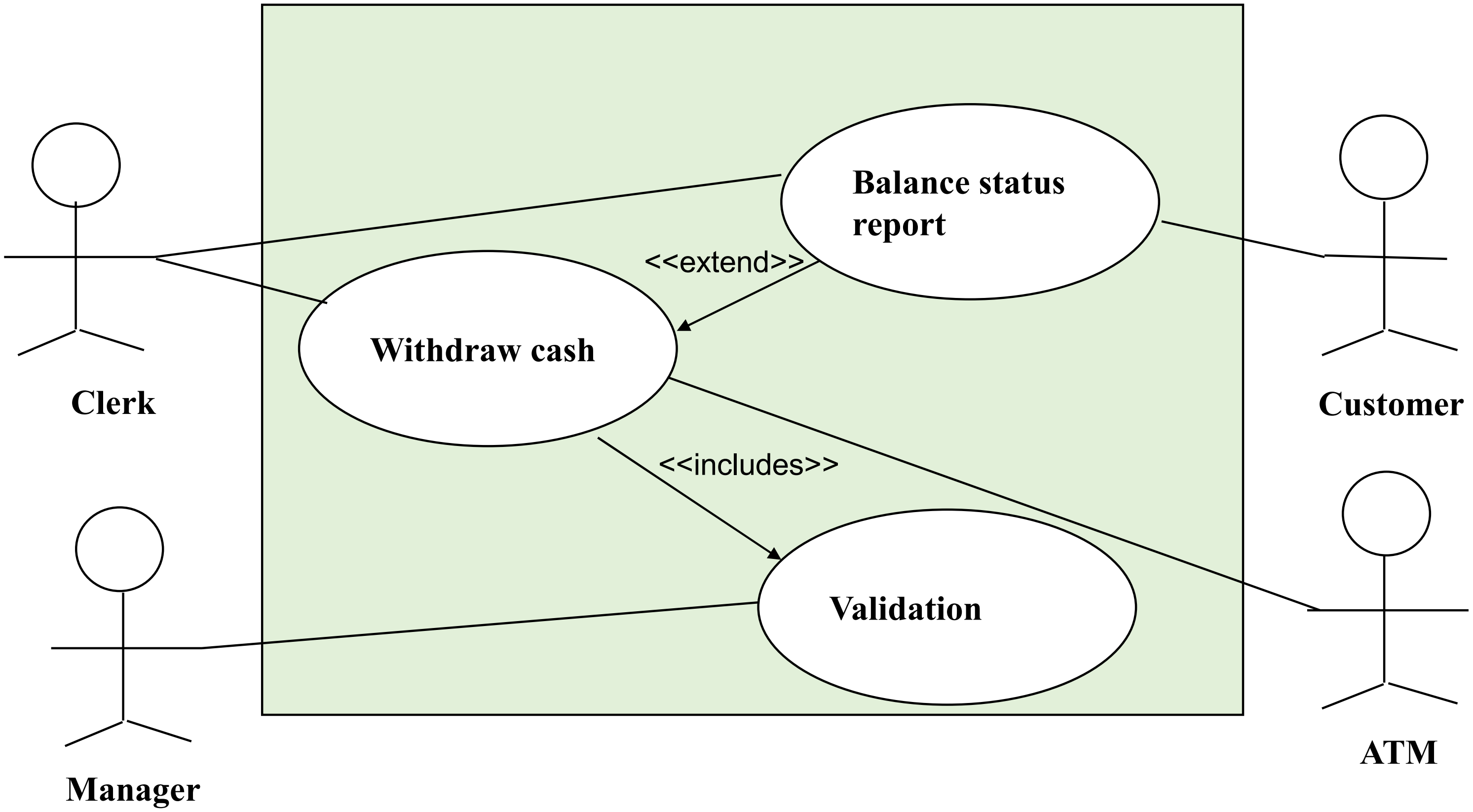
- Use *include* when two or more separate use cases are repeated and to avoid repetition.

- Use *generalization* when describing a variation on normal behavior and wish to describe it casually.

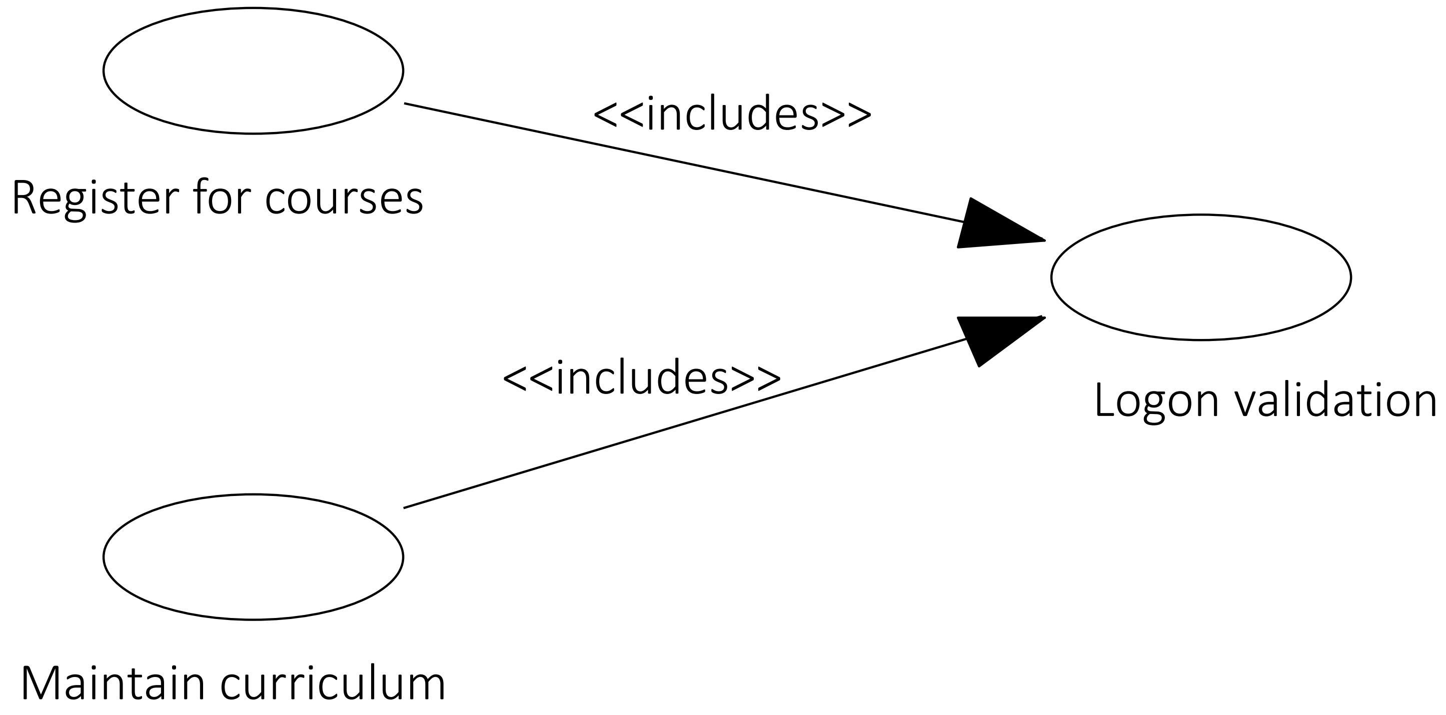
- Use *extend* when describing a variation on normal behavior and wish to use the more controlled form, declaring extension points in base use case.

USE CASE diagram example

Use case diagram for the shown functionality.


Balance status report
<<extend>>
Withdraw cash
Clerk
Customer
<<includes>>
Validation
ATM
Manager

USE CASE diagram example



Register for courses

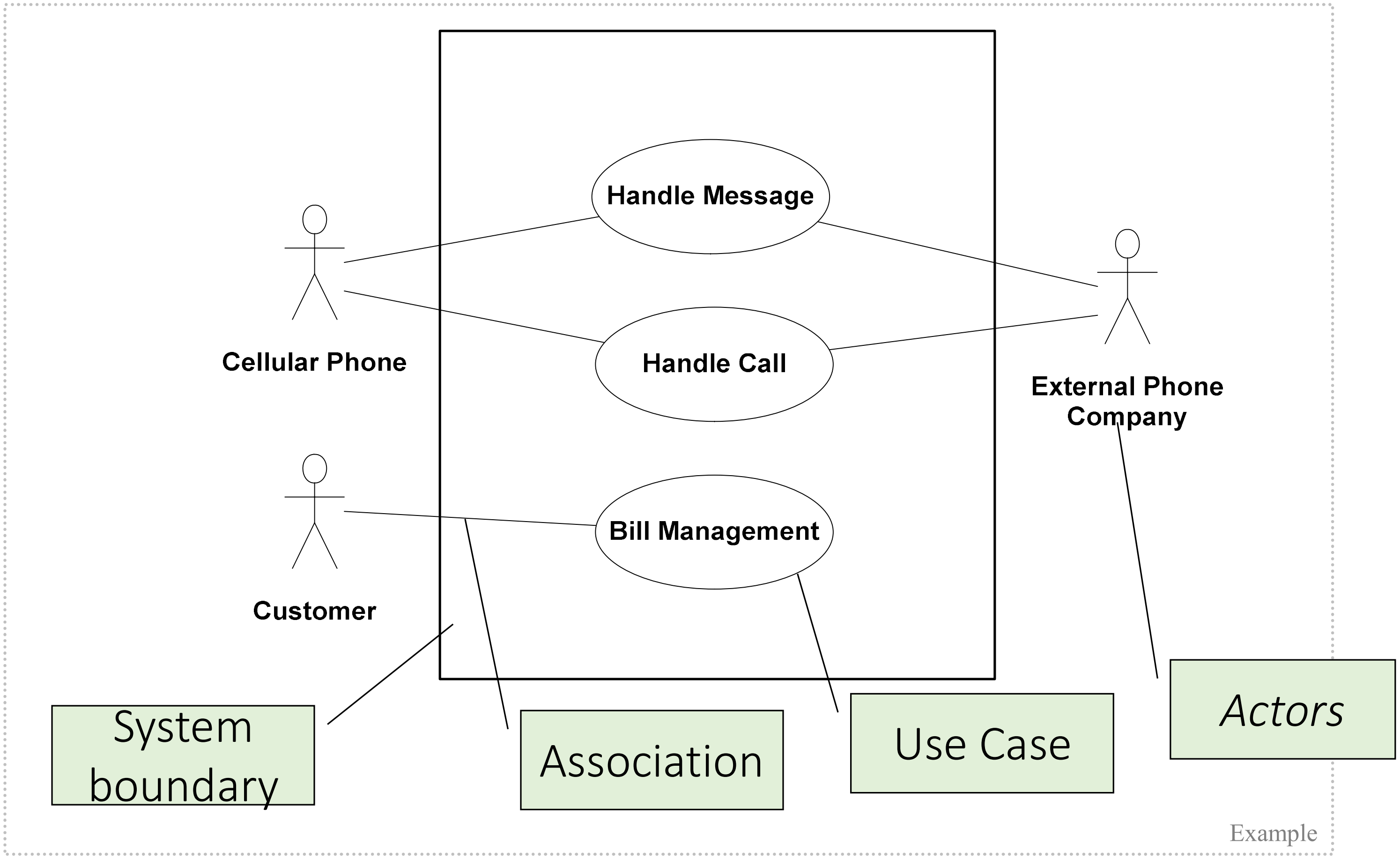
<<includes>>

<<includes>>

Logon validation

Maintain curriculum

A Simple Example


Handle Message
Cellular Phone
Handle Call
External Phone Company
Customer
Bill Management
System boundary
Association
Use Case
Actors
Example

Structure of a Use Case Specification

Name

Actors

Trigger

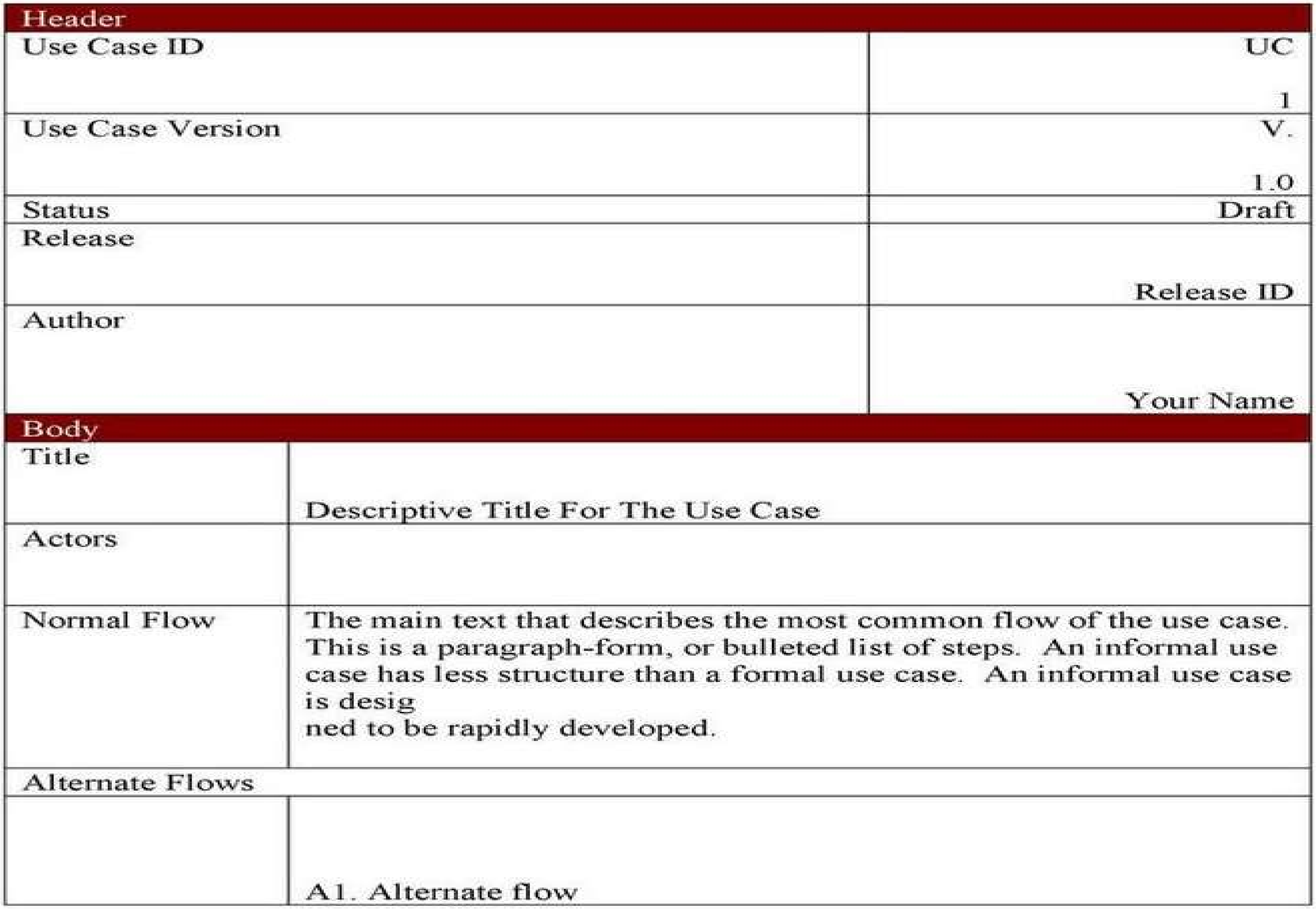
Preconditions

Post conditions

Success Scenario

Alternatives flows

Structure of a Use Case Specification



Header	
Use Case ID	UC 1
Use Case Version	V. 1.0
Status	Draft
Release	Release ID
Author	Your Name
Body	
Title	Descriptive Title For The Use Case
Actors	
Normal Flow	The main text that describes the most common flow of the use case. This is a paragraph-form, or bulleted list of steps. An informal use case has less structure than a formal use case. An informal use case is designed to be rapidly developed.
Alternate Flows	
	A1. Alternate flow

Preconditions

- What the system needs to be true before running the use-case.
- Examples
  - User account exists
  - User has enough money in her account
  - There is enough disk space

Post-Conditions

- A post-condition is the outcome of the use-case.
- Examples
  - Money was transferred to the user account
  - User is logged in
  - The file is saved to the hard-disk

Steps in creating a Use-Case

1. Identify use-cases (find major functions)

2. Draw the system boundary

3. Place use-cases on the diagram (6-8 use-cases per diagram)

4. Identify the actors

5. Add associations

In addition…

- Text documents to document use-cases since use-case diagram does not describe how those use cases are carried out by the actors.
- When describing the use-case,
- you should focus on its external behavior
- – how it interacts with the actors,
- rather than how the use case is performed inside the system

Steps in Documentation

1. *Create a use case template that has areas labeled Basic Course and Alternative Courses*.

2. *Ask "What happens?"*

3. *Ask "And then what happens?"*

4. *Ask, "What else can happen?*

Documentation examples

- Patient Admission
- Rent a car
- Search by Author

Includes statement

- When one use-case needs to use another use-case

- Two types
  - Internal – only other use-cases reference
  - Both external actors and internal use-cases reference

Examples

- Order Entry
- Class registration

Benefits of Use-Case Models

- Used to communicate with the end users and domain experts
  - Provides buy-in at an early stage of system development
  - Insures a mutual understanding of the requirements

- Used to identify
  - Who interacts with the system and what the system should do
  - The interfaces the system should have

- Used to verify
  - All requirements have been captured
  - The development team understands the requirements

First Cut at Application Requirements

- Use Cases
- Amount to ‘placeholders’ for expanded use cases (to come).
- Contain name and short description of the interaction; contains initiating actors.
    - Verb…objects.
- Typically, we do NOT know all the details.
- Want to capture the ‘architecturally significant’ use cases… (What are these???)
- Trying to identify key functions / risks / interactions at a global level.

Use Cases

- Ensure façade use cases are *user-centric* and NOT *technology-centric.* Why? What does this mean?

- Again, these are the WHATS that the users want! These are the ‘behaviors’ of the system.

- Express in terms the users understand!

- Involve various sources early and frequently

    - Many different stakeholders with diverse ‘takes’ on the application…

Use Case Index

- This is a Table of Contents for your Use Cases (that will be developed). Single Sheet; table in Word
- For each Use Case in the index (include a designator (like UC1, UC2, …),
- followed by the use case name, short description (two or three sentences),
- actors, level (façade, filled, focused) date last updated (for now).
- You may also write a 'short user story' describing the use case objectives. If so, this must be very short!!
- "Façade use cases show the basic, essential, high-level interactions that this application must support (Constantine 1999)."

Use Case Number	Use Case Name	Description	Actors	Level	Date of Update
1	Upload Document	Customer will be able to upload documents for translation	Customers, Database	Focused	11/13/2006
2	Translate Document	The System verifies that payment has been received from the billing company and proceeds to translate the source document. The language consultant reads and corrects errors in the virtual document.	Billing Company, Language Consultant, Database	Focused	11/13/2006
3	Ship Document	System administrator arranges for shipments to customers. Shipping company ships final product hard copies and reports the status of shipments to the system administrator.	Customer, Shipping company, System Administrator	Focused	11/13/2006
4	Maintain Database	Administrator corrects errors and updates all system databases.	System Administrator, Database	Focused	11/13/2006
5	Update Profile	The customer is able to change their personal information.	Customer, Database	Focused	11/13/2006
6	Change Order	The customer is able to change their order attributes.	Customer, Database	Focused	11/13/2006
7	Retrieve Document	Customer retrieves target documents.	Customer, Database	Focused	11/13/2006
8	**Print Document**	**System directs customers to Printing company. Printing company provides printing options to customers and the information is directed back to the System Administrator.**	**Customer, Printing company, System Administrator**	**Focused**	**11/13/2006**

Façade Use case: Attributes (Rows)

- Name of the use case – short name (verb, object); action words;
- Actor(s) that trigger the use case…
- Level (façade, filled, focused…These refer to levels of granularity.)
- Short Description – two three sentences: at the most! (story?)
- Leave room for the Basic Course of Events…(happy path)
- Pre and Post conditions, and more (see ahead…)
- Trigger (what initiates the use case…. They just don't 'start')
- Business Rules Link (Reference Business Rules in your Use Cases)
- Risk priority (reference your Risk List preferable by number or identifier)
- Link to text on non-functionals (quality metrics; constraints) here, if desired
- Date / author(s)
- We will add attributes like Alternate Paths, Extensions…later)

Use Case Number:	03	
Use Case Name:	Edit Member Account	
Actor (s):	Buyer, Seller	
Maturity:	Focused (note: not façade; has basic course of events too)	
Summary:	This use case is started by a *Buyer* or a *Seller*. It provides the capability for one of these actors to edit their *member profile*.	
Basic Course of Events:	Actor Action	System Response
	1. This use case is started when a *Buyer* or *Seller* elects to edit their *member profile*. Perform S1-Login	2. The System displays the Actor's *member profile* and prompts the Actor to update it.
	3. The Actor updates their *member profile*.	
		4. The System validates the information entered by the Actor. {Validate Information}
		5. The System prompts the Actor for confirmation.
	6. The Actor confirms that the information is correct. {Profile Change}	
		7. The System updates the Actor's *member profile* to the *member repository* and informs the Actor that the information was updated successfully.
	8. This use case concludes when the Actor receives visual confirmation of the update.	

Continuing…

Alternate Paths:	A1 Change Member Profile At {Profile Change} the *Member* indicates that he/she entered incorrect information. The System immediately returns to the step 2.
Exception Paths:	E1 Handle Invalid Information At {Validate Information} if any fields are entered incorrectly. The System indicates the fields that were entered incorrectly and prompts the *Buyer* or *Seller* to make the necessary corrections. The flow of events is resumed at Basic Flow Step 2.
Extension Points:	{Change Profile }, {Validate Information}
Triggers:	The *Buyer* or *Seller* would like to edit their *member profile*.
Assumptions:	The *Buyer* or *Seller* is aware of the steps required to edit their *member profile*.
Preconditions:	The System is functioning properly. Actor already has a Profile stored in the Profile Database???
Post Conditions:	The *member profile* was successfully updated to the *member repository. Actor sent email to confirm changes.*
Reference - Business Rules:	See Business Rules section: 2.3.1 and 2.3.5.
Reference - Risks:	See Risks List sections: 2.1 and 2.4.
Author (s):	*Team3*
Date:	11-04-04

Think About Non-Functional Requirements (Quality Constraints)

- You have seen ‘lists’ of non-functional requirements.
- Be aware of a number of examples of non-functional requirements that may be addressed in your application.
  - Availability, compatibility, data integrity, extensibility, interoperability, maintainability, performance, portability, reliability, robustness, scalability, security, usability / achievability
  - Know these! (be able to identify them…)
- Document these per use case
- Non-functional requirements normally not tied to specific use cases; normally threaded through a number of use cases.
- Capture in a Table
- These normally are found within a Supplementary Specifications Document…

Verbiage in Use Case

- Use Verb-object phrases (stated before…)
  - Sell Houses, Enroll in Course; Maintain Book…
  - Should not be instances of classes
  - Should not be tied to an organization structure, paper forms or computer implementation
  - Refer to Prototype to see actions an actor expects to undertake…(ahead)
  - Use phraseology from Prototype and Domain Model in text.
    - Interface items and domain entries will become objects ahead….
    - Nice technique: Bold items or Italicize items in Use Case Specifications for which there is a glossary entry or a business entity in the Domain Model
    - Have links for key words or entities to domain model / glossary.

Candidate Use Case List

- Ensure each Use Case is ‘in scope.’

- Actors must reflect the roles that people play – not the actual names of people.

- Note: Use Cases will often have multiple actors.

- Again: (repeating…)
  - Actor must provide or receive a value from the system
  - Do not tie your actors to your organization chart.

User Interface Prototype

- Use storyboarding to elicit major, high-level end-user requirements.

- Document the interactions as seen in the storyboards as high level text to identify the Façade Use Cases.

- The details of the interactions (and likely more detailed user interface prototypes) will be used to capture the interactions use cases with more granularity.

- More on prototyping in the next series of slides.

Verb Filter and Noun Filter

- Use strong verbs (‘generate’ ‘calculate’ ‘compute’)
- Use strong, concrete nouns.
- Avoid weak nouns such as data, report, system, form, template, paper…

Façade Filter

- Façade use cases represent the most important interactions between the actors and the system.

- Don’t worry too much about non-functional requirements at this time. Will address more later.

- Façade use cases should be relatively abstract – so that they will cover a number of actual proposed interactions (scenarios).

- No implementation details!

- Façade use cases contain NO basic course of events….

- Only trying to identify significant Use Cases – for their functionality, their actors, pre and post conditions, triggers, and key attributes…

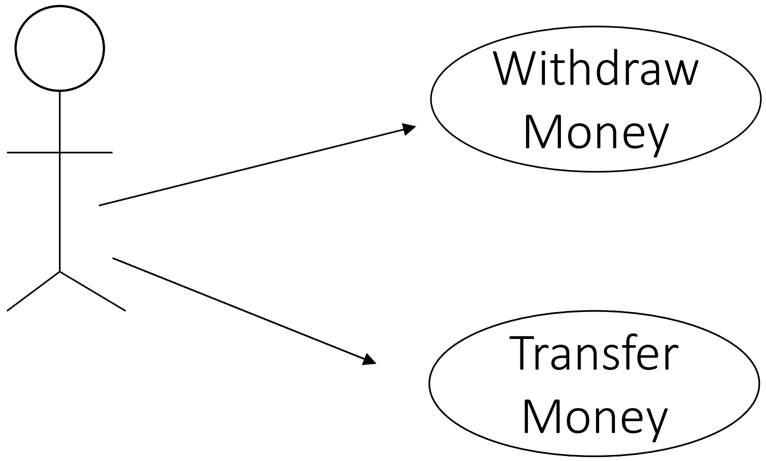
Reviews

- Peer Reviews:
    - Review the use cases carefully.
    - Have a 'SME' available for business domain questions.
    - Have reviews often and informally
- User Reviews:
    - User review of your Façade use cases is critical.
    - Every hour you spend in review could save many hours of work later!!!
    - It is RECOMMEND THAT TEAMS REVIEW EACH OTHER'S USE CASES!!
    - Use Cases capture functional requirements…
    - Sometimes called Requirements Analysis…It is better to keep 'Requirements' and 'Analysis' somewhat separate.

Packaging

- You need to build packages of Use Cases group Use Cases of similar ‘value’ or ‘functionality’

Use Case Diagrams… (a graphical model)


Withdraw Money
Transfer Money

Simply actors and use cases

Can, of course, be much more detailed. Often the relationships are ‘tagged.’

Extending the UML with Stereotyping

- Know we have 'Change' in everything.
- But very few graphics in UML.
- Need to communicate special cases:
  - special classes
  - special kinds of use cases…
  - Extend UML for new 'types'
  - New types of model elements?
  - We often need customization of models for some projects.

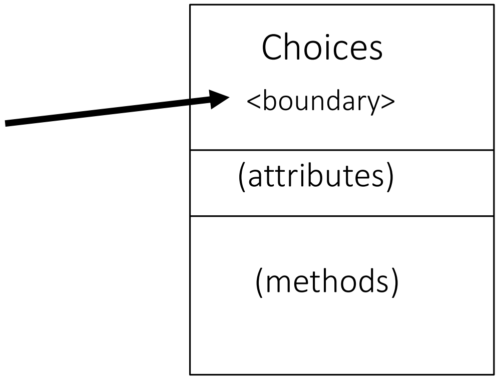
Extending the UML with Stereotyping

- Enter: Stereotyping.
  - Allow us to 'refine' / 'reclassify' 're-specify' all model elements into a more specialized form **rather than** create additional symbols!
  - We might specify a Use Case as a <<mission critical>> or class name with the stereotype: <<boundary>> or <<control>> etc.
  - Indicate that the symbol is still a Use Case – but a 'special one' perhaps in a 'special context.'
- Most common UML stereotyped element is the class.
- Stereotyping makes these different model elements!!!
  - (Incidentally, additional icons can be added, if wanted…)

Examples

A ‘special’ kind of class with special behaviors – a boundary class.

A ‘special’ kind of Association – indicates a use case that supports a more fundamental use case – one that is more significant.

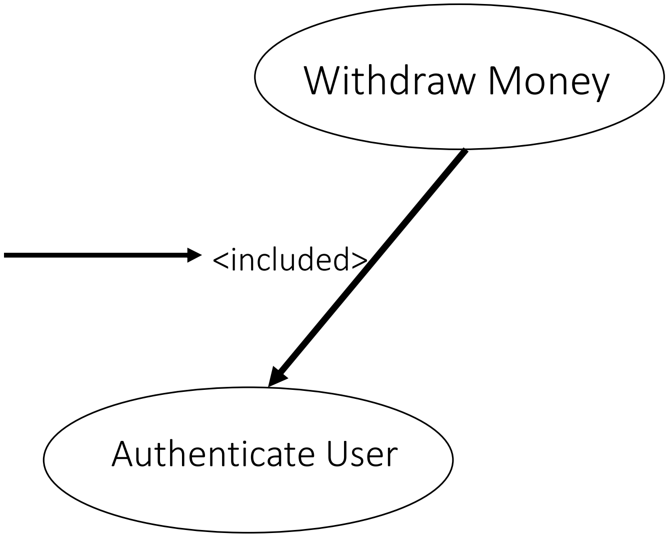


Choices

<boundary>

(attributes)

(methods)



Withdraw Money

<included>

Authenticate User

Stereotypes in Modeling: Built-ins and User-Defined

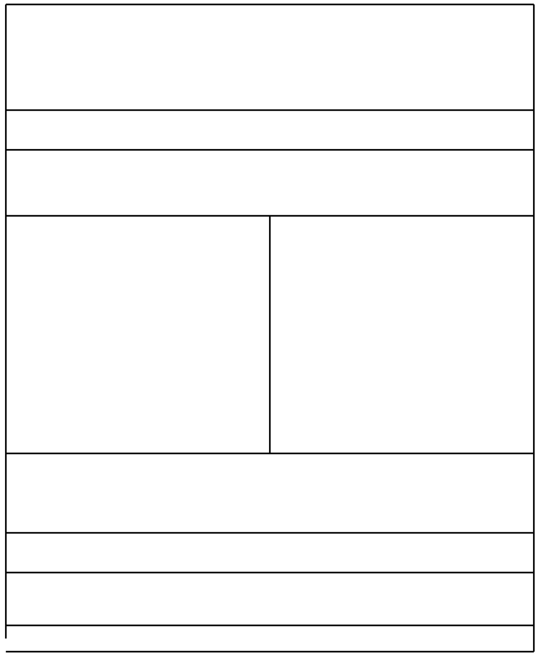
- Stereotypes can be used to ‘increase relevance’ of model elements, such as use cases in requirements gathering.
  - (Much controversy on ‘**extend** use cases’ and ‘**include** use cases’ Much more later: stereotypes: <<includes>> and <<extends>>
- Use Cases are quite commonly stereotyped
  - A <mission critical> use case ‘may’ be specified in a separate document addressing all stereotypes
  - “Stereotyped element” implies that there are ‘special’ criteria.
  - e.g. A use case that is “mission critical” => must be processed within five seconds.

Stereotypes in Modeling: Built-ins and User-Defined

- Classes may also very often stereotyped:
  - <boundary>, <control>, <entity> (as found in Analysis Modeling)
    - A boundary class is a special kind of class that interacts directly with an actor…
- Any UML modeling element may be stereotyped….

Use Case Template (Be aware there are many many formats. Format is not critical. Content is.)

One of many different kind of formats…



Use Cases

- Use Cases – a great tool that helps us to express interactions between system (application) and actors.

- We can see the behaviors of the system.

- We can see the scenarios (instantiation of a use case w/data)

- Meaningful to customer who is concerned with the 'whats' of the system to see how system and actor interact with each other.

- Successful development of Use Cases requires an understanding of the goals of each of the use cases, which, in turn, is developed from identified, required functionality – namely Features.

Goals: Use Cases capture ‘Whats’

- “Context-free” use (that is, no design constraints)
- Excellent for requirements gathering / modeling; analyzing and capturing desired user interactions
- Numbers of Use Cases?
  - 70-80 for very large system?
  - Medium sized – 20 to 50;
  - Small systems – fewer. Often 6 to 8 may be sufficient.
- Smaller number forces abstraction. Desirable.
- There is no ‘one size / number’ fits all. But there are guidelines in their development! (Coming)

Goals of Use Cases

- Interactions must present a value to actor.
    - actor may be an accounting system; general ledger system; person; customer; a relay; or thermostat…
        - person, device, external system/database/file.
        - Actors are external to the system.

Goals of Use Cases

- Use Cases are black box representations
  - Do not show implementation-specific language
  - Do not want to imply how use case will be realized.
  - Do not include language such as: people, interface widgets (buttons, menus…), IFTHENELSE statements, pseudo-code, more…
  - Are written in language of the application domain.
  - Are ‘abstractions’ of detailed interactions.
  - Capture stories addressing functional requirements…

BITS Pilani
Pilani Campus

SOFTWARE DEVELOPMENT LIFECYCLE

Use cases Include & Extend relationship

By Prof A R Rahman

Most efficient way to write use cases

- Use cases model requirements, they are highly dynamic by nature.
- The more we examine a scenario, the more we learn and the more things change.
- To further complicate the issue, changes to one use case can lead to changes in others.
- Therefore, we want a flexible, highly efficient method for writing use cases that maximizes stakeholder value and minimizes risk early in the project and minimizes costly rework later.
- An iterative, breadth-before-depth approach is best.
- This breadth-first approach involves two aspects: finding and outlining use cases and detailing individual use cases.

Finding and outlining use cases

- Use cases exist in sets, and the relationships between the various use cases and Actors are important.
- As you learn more about the Actors, you also learn more about the system's boundaries and transactions.
- Likewise, as you learn more about the system's transactions, you learn more about its Actors.
- Therefore, it is more efficient to capture and outline the primary use cases before detailing individual use cases.
- This way, you can identify and understand the importance and risk associated with each use case before committing time to detailing them.
- This aspect of the breadth-before-depth approach is embodied in the process by the explicit separation of two tasks Find and Outline Requirements and Detail Requirements.

Detailing individual use cases

- Similarly, it makes sense to write each individual use case iteratively.
- Start by detailing the main scenario.
- As we do this, we can identify various alternative and error flows that the use case might follow, and then evaluate, rearrange, or eliminate them and prioritize them before detailing the surviving scenarios, so you can focus your effort in the right place.
- The level of detail that you capture depends on several factors.

It does not describe:

- The GUI
- Technical details of hardware or software
- Design issues
- Describe the flow of events for the main scenario in the form:

1. The use case starts when <Actor name> <does something>.
2. The system <does something in response>.
3. The <Actor name> does something else.

Detail the basic flow of events (main scenario)

- As a starting point, use the step-by-step description of the main scenario that you created during Task: Find and Outline Requirements.
- Then, gradually add details to this scenario, describing what the use case does, not how to solve problems internal to the system.
- A flow of events description covers:
- How and when the use case starts
- When the use case interacts with the Actors and what data they exchange
- When the use case uses data stored in the system or stores data in the system
- How and when the use case ends

Detail alternate flows

- A use case consists of a number of scenarios, each representing specific instances of the use case that correspond to specific inputs from the Actor or to specific conditions in the environment.

- Each scenario describes alternate ways that the system behaves, or it may describe failure or exception cases.

- Review the list of alternative flows that you captured during Task: Find and Outline Requirements.

- As you detail the main scenario, you may identify additional alternate flows by asking these questions:

Detail alternate flows

- Are there different options available, depending on input from the Actor?
- (Example: If the Actor enters an invalid PIN number while accessing an ATM.)
- What business rules may come into play?
- (Example: The Actor requests more money from the ATM than is available in her account.)
- What could go wrong? (Example: No network connection available when required to perform a transaction.)

It is best to develop these scenarios iteratively, as well.

1. Begin by identifying them.

2. Examine each possible scenario to determine whether it is relevant, that it can actually happen, and that it is distinct from other scenarios.

3. Eliminate redundant or unnecessary scenarios.

4. Prioritize the remaining scenarios and start detailing the more important ones.

It is best to develop these scenarios iteratively, as well.

- In addition to the detailing the flow of events of each alternative flow in the form described above, each alternative flow should describe:

- Where the alternative flow can be inserted in the basic flow of events

- The condition that needs to be fulfilled for the alternative behavior to start

- How and where the basic flow of events is resumed, or how the use case ends

Structure the use case

- It is useful to structure the use case according to scenarios.

- This helps both to simplify communication and maintenance and to permit the use cases to be implemented iteratively.

- Name and describe each key scenario so that these may be prioritized, assigned to an iteration and an individual, and referenced from the Work Items List for implementation.

Structure the use case

- In addition to structuring the use cases according to scenarios, it is often useful to structure the scenarios themselves into subflows.

- This provides an additional level of granularity for planning work and tracking progress.

- Unless a subflow involves only a minor part of the complete flow of events (which can be described in the body of the text), describe each subflow in a separate section of the Flow of Events section.

- Subflows that should be in a separate section.

Subflows that occupy a large segment of a given flow of events.

- Exceptional and alternative flows of events.

- This helps the use case's basic flow of events to stand out more clearly.

- Any subflow that can be run at several intervals in the same flow of events.

- If there are subflows identified that are common to multiple use cases, you can re-factor your use-case model to include these.

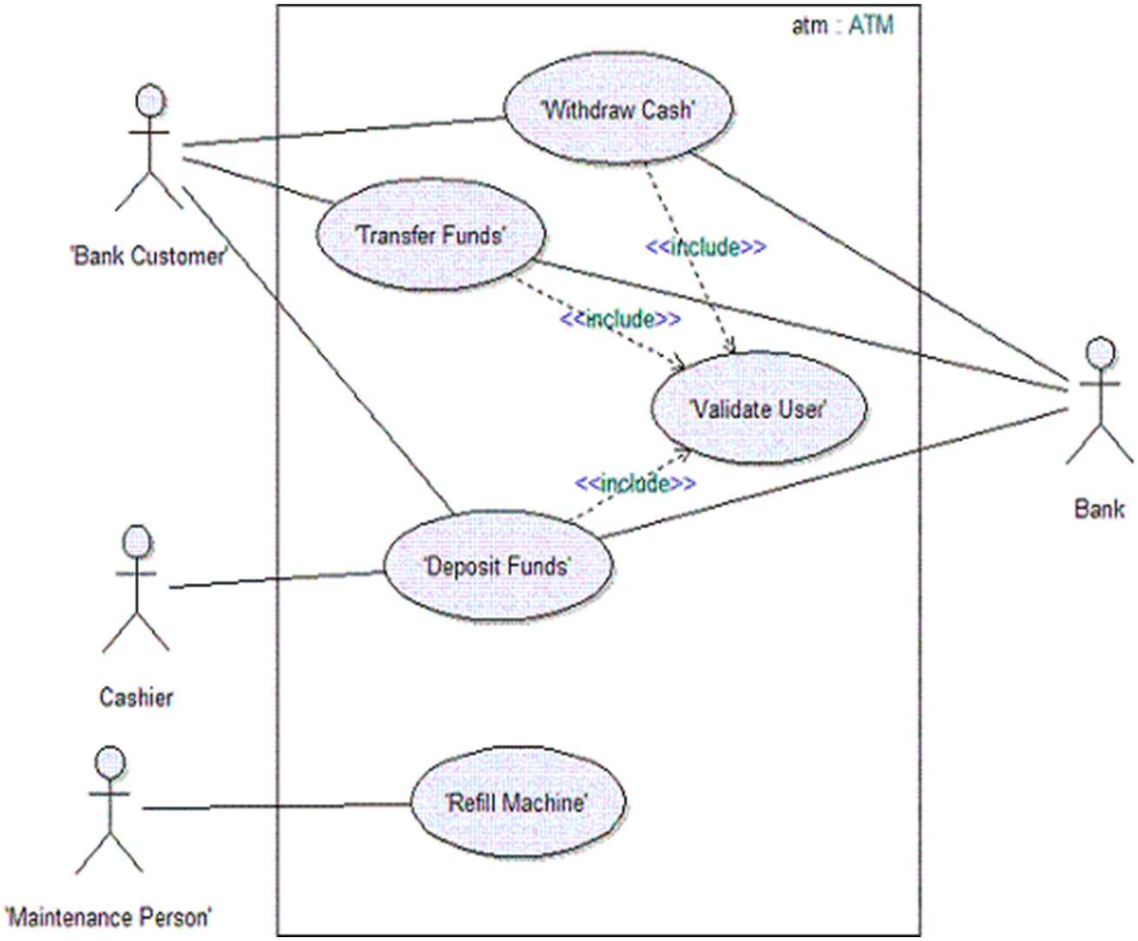
Subflows that occupy a large segment of a given flow of events.

- See Concept: Use-Case Model for more information on the <<include>> dependency.

- For more information on structuring the use case, see the "Flow of Events - Structure" section in Concept: Use Case.

- The Template: Use-Case Specification provides a suggested structure for the use case specification.

Example


atm : ATM
'Withdraw Cash'
'Transfer Funds'
<<include>>
<<include>>
'Validate User'
<<include>>
'Deposit Funds'
'Refill Machine'
'Bank Customer'
Cashier
'Maintenance Person'
Bank

The <<extends>> Relationship

- **Extend** is a directed relationship that specifies how and when the behavior defined in usually supplementary (optional) **extending use case** can be inserted into the behavior defined in the **extended use case**.

- **Extended** use case is meaningful on its own, it is **independent** of the extending use case

- **Extending** use case typically defines **optional** behavior that is not necessarily meaningful by itself.

The <<extends>> Relationship

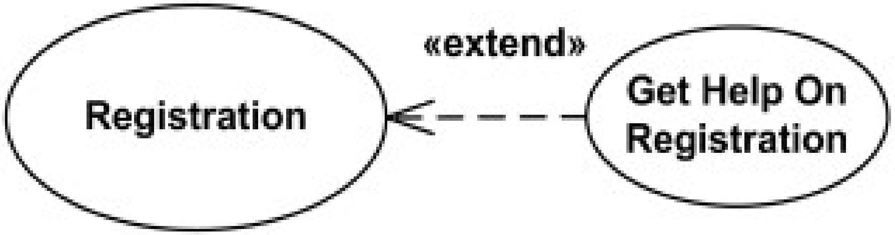
- The extend relationship is owned by the extending use case.

- The same extending use case can extend more than one use case, and extending use case may itself be extended.

- The extension takes place at one or more extension points defined in the extended use case

The <<extends>> Relationship

- Extend relationship is shown as a dashed line with an open arrowhead directed from the extending use case to the extended (base) use case. The arrow is labeled with the keyword «extend».


«extend»
Registration
Get Help On Registration

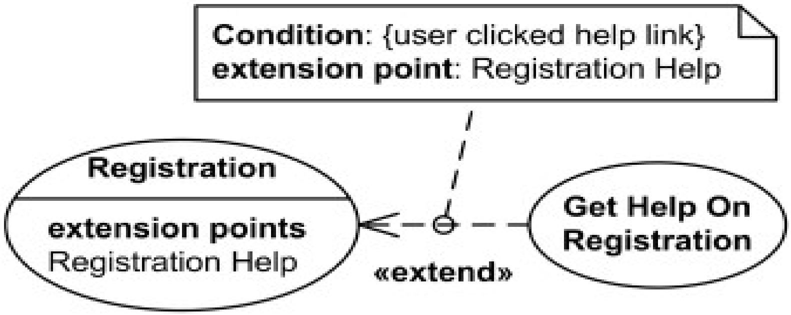
Registration use case is meaningful on its own.

It could be extended with optional Get Help On

Registration use case

The <<extends>> Relationship

- The condition of the extend relationship as well as the references to the extension points are optionally shown in a comment note attached to the corresponding extend relationship


Condition: {user clicked help link}
extension point: Registration Help
Registration
extension points
Registration Help
«extend»
Get Help On Registration

Registration use case is conditionally extended by Get Help On Registration use case in extension point Registration Help

The <<extends>> Relationship

- An **extension point** is a **feature** of a use case which identifies (references) a point in the behavior of the use case where that behavior can be extended by some other (extending) use case, as specified by extend relationship.

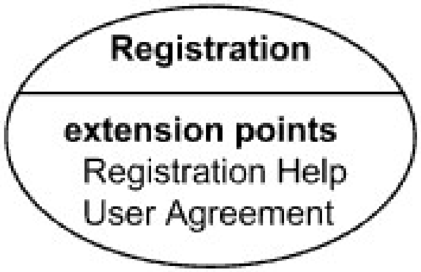
- Extension points may be shown in a compartment of the use case oval symbol under the heading **extension points**.

- Each extension point must have a **name**, unique within a use case.

The <<extends>> Relationship

- Extension points are shown as a text string according to the syntax:

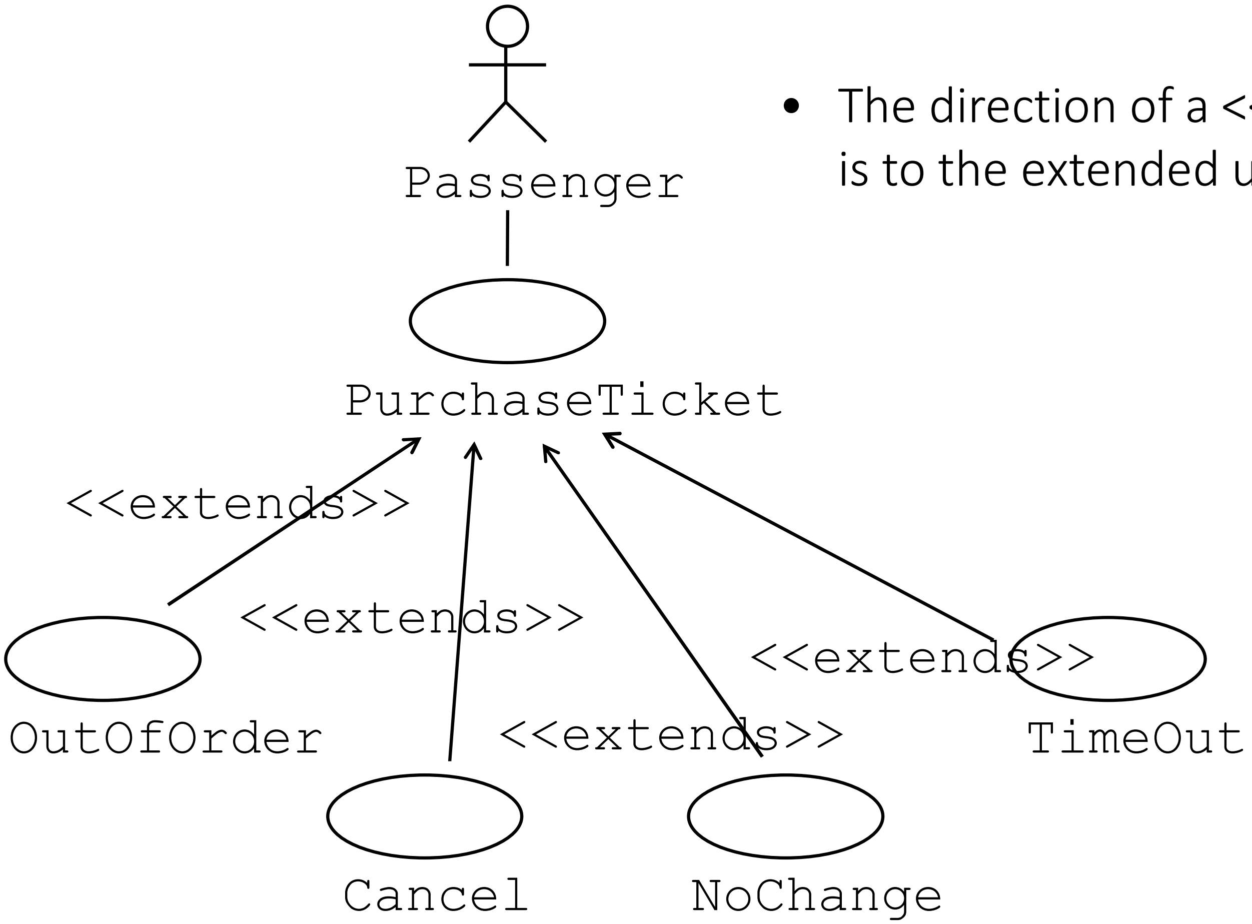
  extension point ::= name [: explanation]


Registration
extension points
Registration Help
User Agreement

Registration use case with extension points

Registration Help and User Agreement

The `<<extends>>` Relationship


Passenger
PurchaseTicket
<<extends>>
<<extends>>
<<extends>>
<<extends>>
OutOfOrder
Cancel
NoChange
TimeOut

- The direction of a <<extends>> relationship is to the extended use case

The <<includes>> Relationship

- An include relationship is a directed relationship between two use cases when required, not optional, behavior of the included use case is inserted into the behavior of the including (base) use case.

- Including use case depends on the addition of the included use case.

- Including use cases are usually not complete by themselves and require the included use cases

The <<includes>> Relationship

- The include relationship could be used:
  - when there are common parts of the behavior of two or more use cases,
  - to simplify large use case by splitting it into several use cases.

- Included use cases are required, not optional for the inclusion.

- Execution of the included use case is analogous to a subroutine call or macro command in programming.

The <<includes>> Relationship

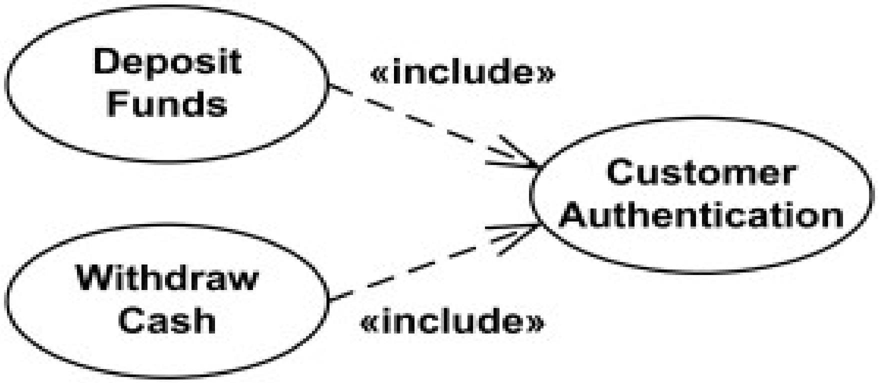
- All of the behavior of the included use case is executed at a single location in the including use case before execution of the including use case is resumed.

- Include relationship between use cases is shown by a dashed arrow with an open arrowhead from the including (base) use case to the included (common part) use case.

- The arrow is labeled with the keyword «include».

The <<includes>> Relationship

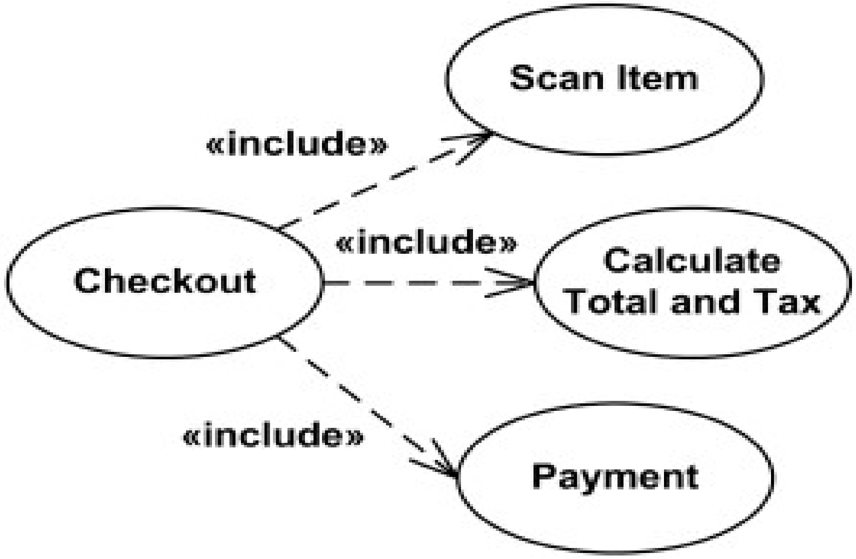
- When two or more use cases have some common behavior, this common part could be extracted into a separate use case to be included back by the use cases with include.


Deposit Funds
«include»
Customer Authentication
Withdraw Cash
«include»

Both Deposit Funds and Withdraw Cash use cases require (include) Customer Authentication use case.

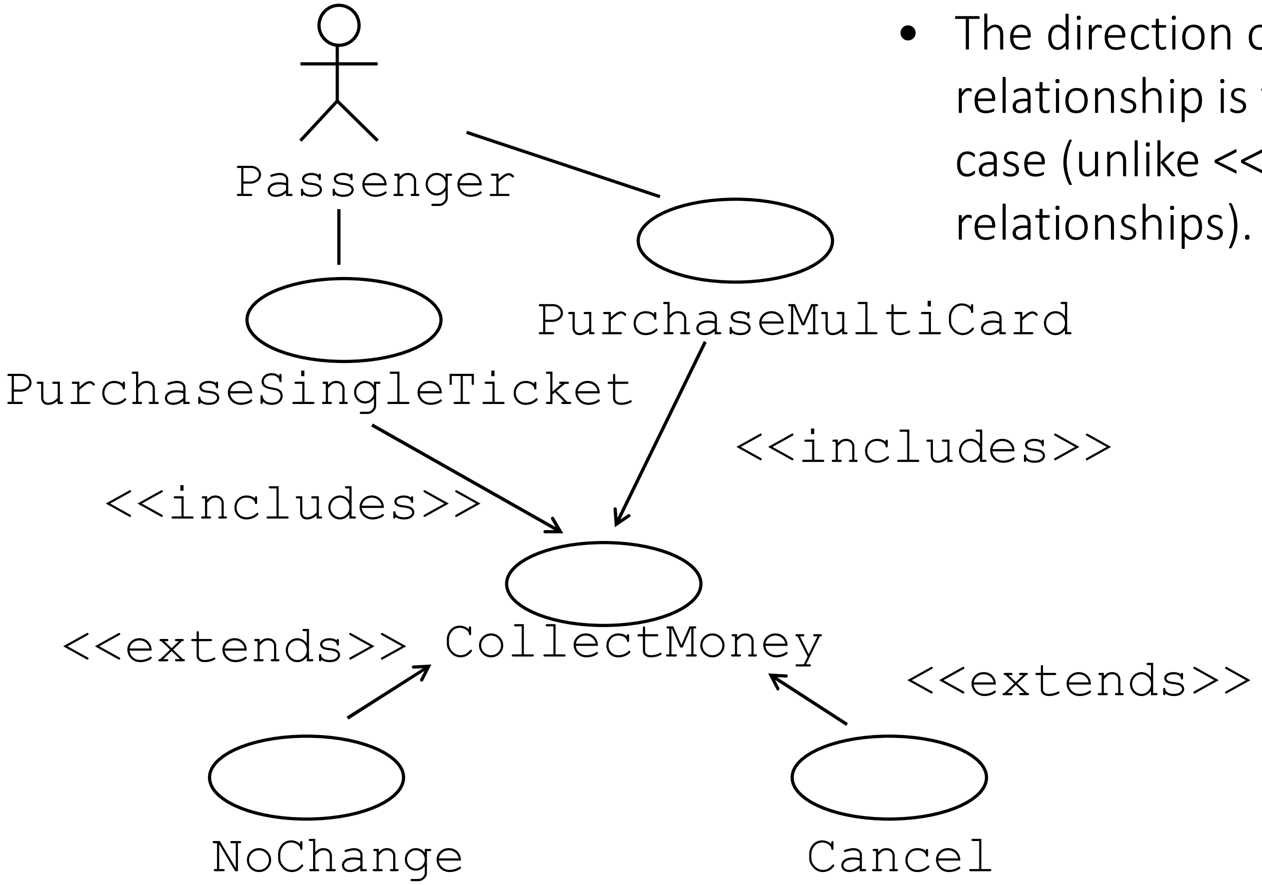
The <<includes>> Relationship

- Large and complex use case could be simplified by splitting it into several use cases each describing some logical unit of behavior.
- Note that including use case becomes incomplete by itself and requires included use cases to be complete.


Scan Item
«include»
Checkout
«include»
Calculate Total and Tax
«include»
Payment

Checkout use case includes (possibly several) Scan Item, Calculate Total and Tax, and Payment use cases

The <<includes>> Relationship


Passenger
PurchaseMultiCard
PurchaseSingleTicket
<<includes>>
<<includes>>
CollectMoney
<<extends>>
<<extends>>
NoChange
Cancel

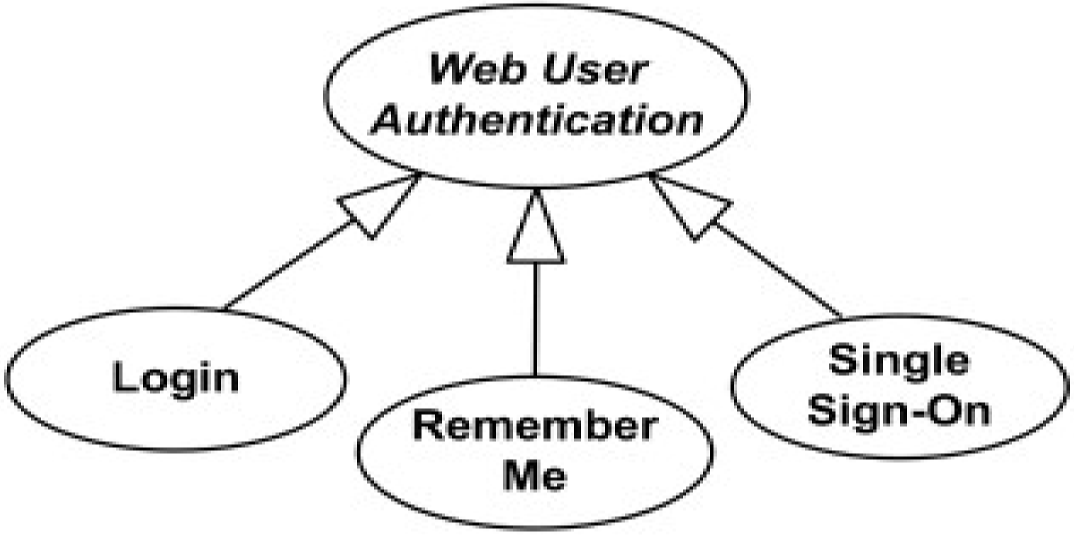
- The direction of a <<includes>> relationship is to the using use case (unlike <<extends>> relationships).

Generalization Between Use Cases

- Generalization between use cases is similar to generalization between classes – child use case inherits properties and behavior of the parent use case and may override the behavior of the parent.

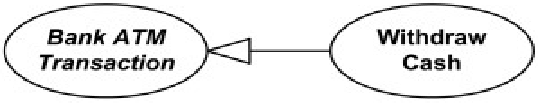
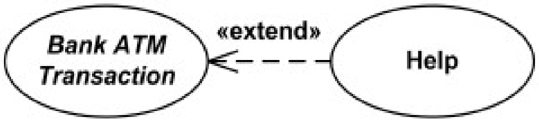
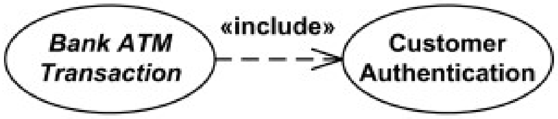
- Generalization is shown as a solid directed line with a large hollow triangle arrowhead, the same as for generalization between classifiers, directed from the more specific use case to the general use case

Generalization Between Use Cases


Web User Authentication
Login
Remember Me
Single Sign-On

Web User Authentication use case is abstract use case specialized by Login, Remember Me and Single Sign-On use cases

Use Case Relationships Compared

Generalization	Extend	Include
 Bank ATM Transaction Withdraw Cash	 Bank ATM Transaction «extend» Help	 Bank ATM Transaction «include» Customer Authentication
Base use case could be abstract use case (incomplete) or concrete (complete).	Base use case is complete (concrete) by itself, defined independently.	Base use case is incomplete (abstract use case).
Specialized use case is required, not optional, if base use case is abstract.	Extending use case is optional, supplementary.	Included use case required, not optional.
No explicit location to use specialization.	Has at least one explicit extension location.	No explicit inclusion location but is included at some location.
No explicit condition to use specialization.	Could have optional extension condition.	No explicit inclusion condition.

Misuse of Relationship

- The <<include>> relationship means that the flow of that use case is always included.

- But a <<extend>> relationship means that the flow of the extending use case is only included under specific conditions, which must be specified as the extension point of the use case being extended.

- Many people get tripped up by <<include>> and <<extend>> relationships between use cases

Tips for getting for Include & Extends

1. Make sure you have got the right kind of relationship:

- <<include>> means always included , but <<extend>> means conditionally included .

2. Make sure you have got the arrows going the right way.

- <<include>> should point towards the use case being included.

<<extend>> should point towards the use case(s) being extended (and not the extending use case).

Exercise

- I am the manager of a theatre.

- I want to create an automated movie ticket machine.

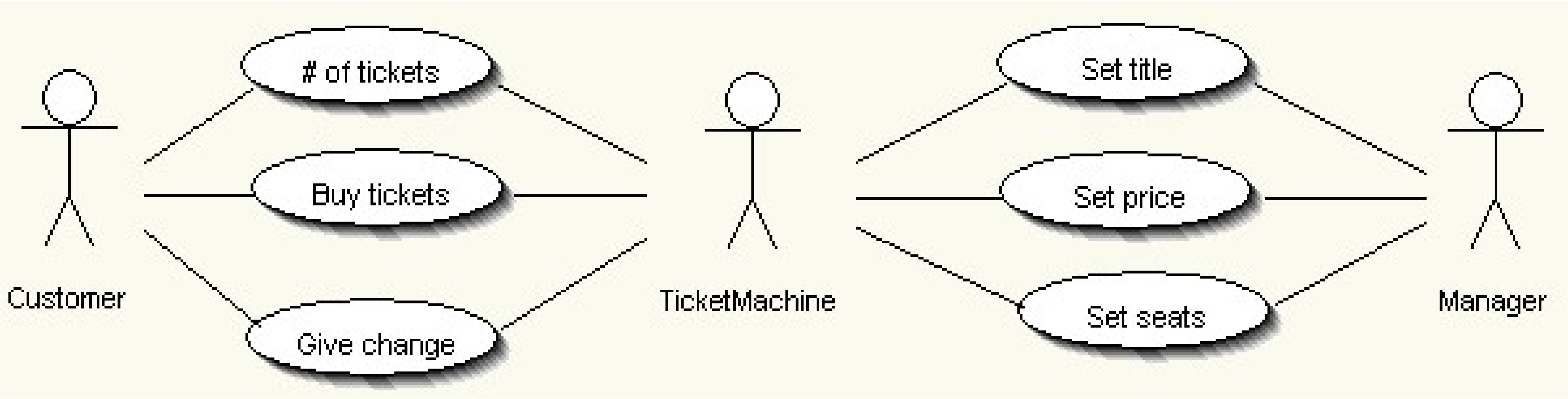
- You are analysts who need to describe
  what the customer wants as a set of use cases

- Simplifying assumptions:
  - One movie showing at a time
  - Movie time is same every day, only one time, same price
  - Only manager can change/add movie
  - Customer can only buy tickets

- Who or what are the actors?

- What are the use cases (goals of actors)?

Use case diagram for Movie Ticket Machine


Customer
# of tickets
Buy tickets
Give change
TicketMachine
Set title
Set price
Set seats
Manager

- Why are there three Actors?
- Why three use cases for Customer?
- Which use cases look easy to write?

Use cases for Manager

Use case: Set title
Actors: Manager, Machine
1. Manager requests a change of movie title
2. Machine asks manager for new movie title
3. Manager enters movie title

Use case: Set price
Actors: Manager, Machine
1. Manager requests a change of ticket price
2. Machine asks manager for new price for movie title
3. Manager enters ticket price
Alternatives: Invalid price
If manager enters price below $1 or greater than $10
3a. Machine asks manager to reenter price

- Use case: Set seats
- Actors: Manager, Machine
- 1. Manager requests a change in number of seats
- 2. Machine asks manager for number of seats in theatre
- 3. Manager enters number of seats
- Alternatives: Invalid number of seats
- If manager enters number less than 20 or greater than 999
- 3a. Machine asks manager to reenter number of seats

Use cases for Customer

Use case: # of tickets

Actors: Customer, Machine

1. Customer enters number of tickets
2. Machine displays total balance due

Alternative: Customer wants zero tickets

At step 1, customer enters zero tickets

1a. Display thank you message

1b. Set balance to $0.0

Use case: Return change to customer

Actors: Customer, Machine

1. Customer requests change
2. Machine dispenses money
3. Machine updates customer balance

Use case: Buy tickets

Actors: Customer, Machine

1. Customer requests tickets
2. Machine tells customer to put balance due in money slot
3. Customer enters money in money slot
4. Machine updates customer balance
5. Customer requests tickets
6. Machine prints tickets
7. Machine updates number of seats

Alternative: Insufficient seats

At step 1, if number of tickets requested is less than available seats,

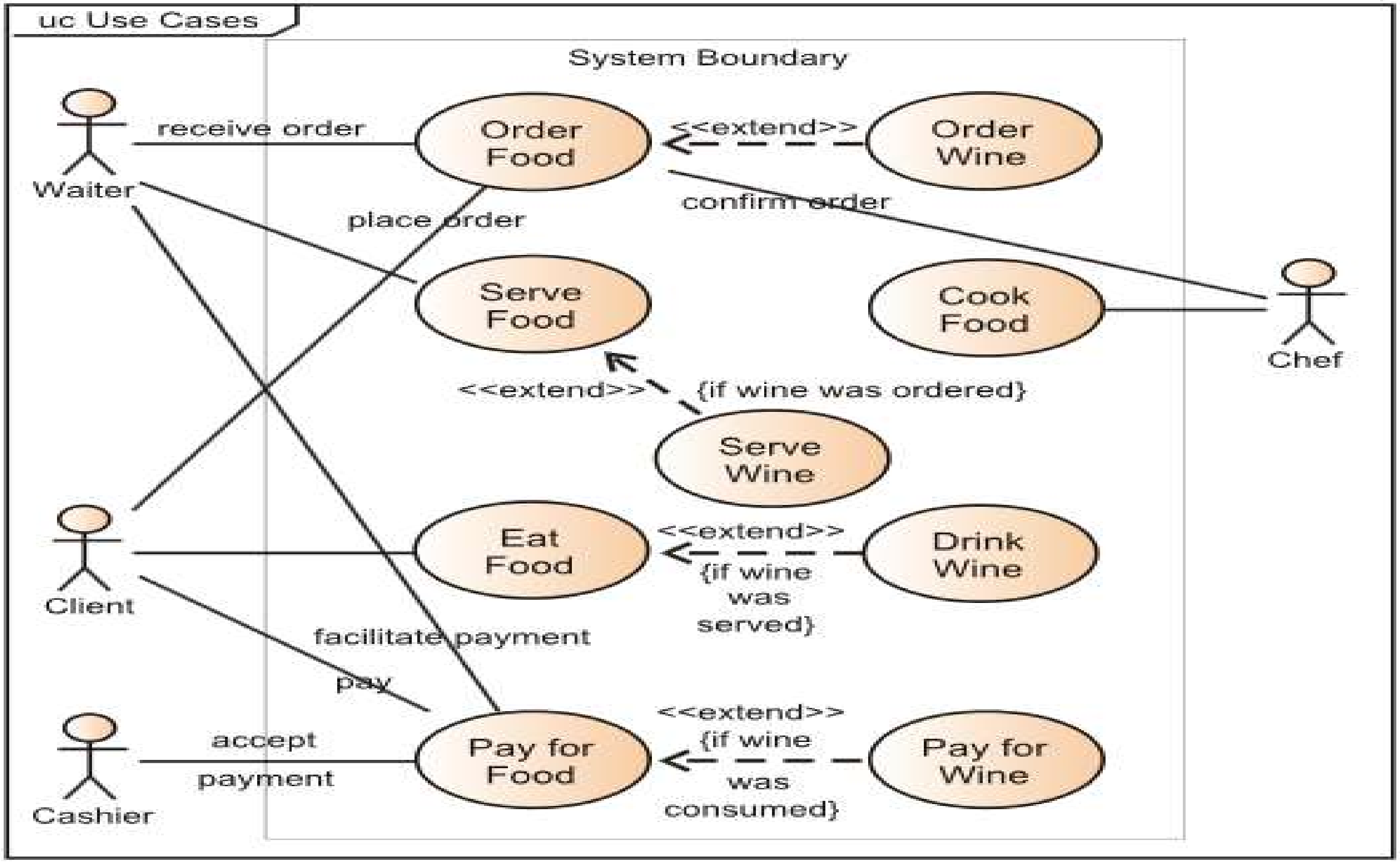
1a. Display message and end use case

Alternative: Insufficient funds

At step 5, if money entered < total cost,

5a. Display insufficient amount entered

5b. Go to step 3


uc Use Cases
System Boundary
Waiter
receive order
Order Food
<<extend>>
Order Wine
confirm order
place order
Serve Food
Cook Food
Chef
<<extend>>
{if wine was ordered}
Serve Wine
Client
Eat Food
<<extend>>
{if wine was served}
Drink Wine
facilitate payment
pay
Cashier
accept payment
Pay for Food
<<extend>>
{if wine was consumed}
Pay for Wine

Case Study: Course Registration Problem Statement

- The university will require a new Course Registration system.

- At the beginning of each semester, Professors at the university will register the courses that they are going to teach with the system.

- The system then will allow students to request a course catalogue containing a list of course offerings for the coming semester.

- Information about each course such as Professor, department and prerequisites are included.

- The student then can select four-course offering.

Case Study: Course Registration Problem Statement

- Once the registration process is completed for a student, the registration system sends information to the billing system so the student can be billed for the semester.

- For each semester, there is a period of time that students can change their schedule through adding or dropping courses.

- Professors must be able to access the system to indicate which courses they will be teaching and to see which students signed up for their course offerings

Practice: Find the Actors

- In the Course Registration System Requirements document, read the Problem Statement for the Course Registration case study.

- As a group, identify the following:
  - Actors
  - Description of the actor

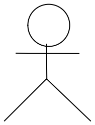
Finding Use Cases: Focus on the Actor

- The system exists only for its users

- Use cases should be based on the user’s needs

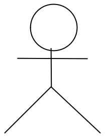
- Answer the following questions to find use cases.
  - For each actor you have identified, what are the tasks the system would be involved in?
  - Does the actor need to be informed about certain occurrences in the system?
  - Will the actor need to inform the system about sudden, external changes?
  - What information must be modified or created in the system?

Practice: Solution



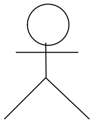
Billing System

The external system responsible for student billing



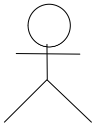
Student

A person who is registered to take courses at the University



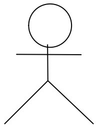
Professor

A person who is teaching classes at the University



Course Catalog

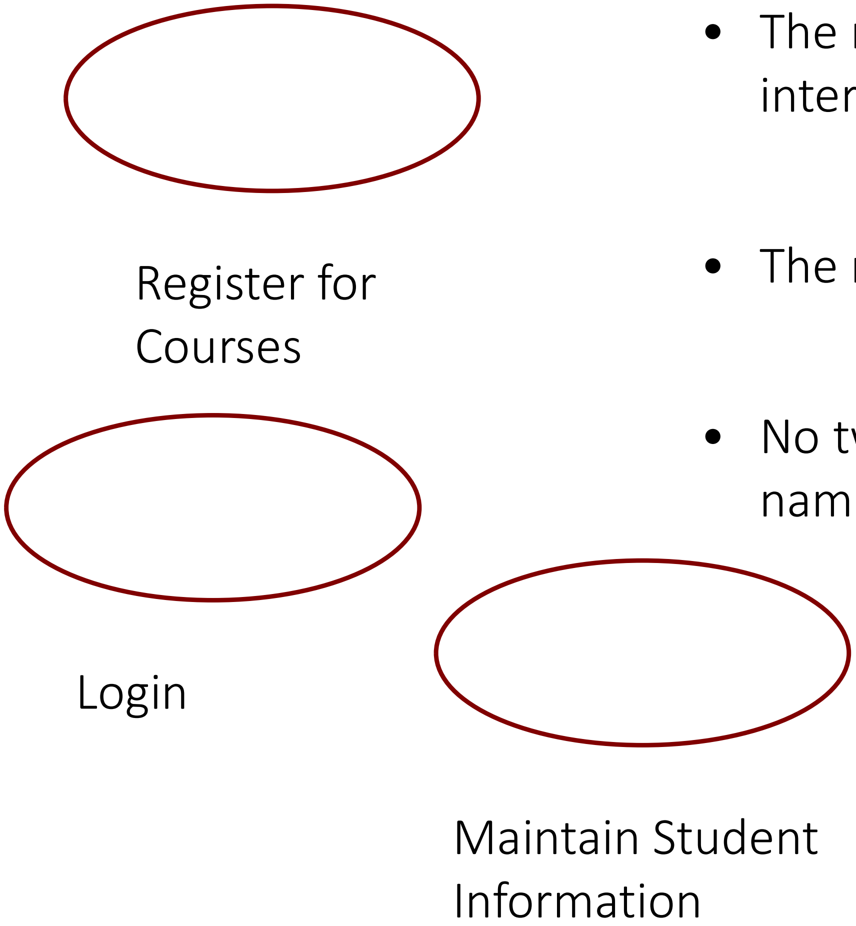
The unabridged catalog of all courses offered by the University



Registrar

The person who is responsible for the maintenance of the course registration system

Name Use Cases



Register for Courses

Login

Maintain Student Information

- The name indicates what is achieved by its interactions with the actor(s).
- The name may be several words in length.
- No two use cases should have the same name.

Practice: Solution

Register for Courses

View Report Card

Login

Select Courses To Teach

Submit Grades

Maintain Professor Information

Maintain Student Information

Close Registration

Use Case Description: Change Flight

- Actors: traveler, client account db, airline reservation system

- Preconditions:
  - Traveler has logged on to the system and selected ‘change flight itinerary’ option

- Basic course
  - System retrieves traveler’s account and flight itinerary from client account database
  - System asks traveler to select itinerary segment she wants to change; traveler selects itinerary segment.
  - System asks traveler for new departure and destination information; traveler provides information.
  - If flights are available then
  - …
  - System displays transaction summary.

- Alternative courses
  - If no flights are available then …

Step Wise Refinement of Use Case Model

The basic steps to generate use cases model for each business process area:

Step 1: Confirm actors and goals.

- Have all actors and their goals been identified?
- Which actors can be generalized (combined)?
- Which goals are potential use cases?

Step 2: Develop an outline of the use case(s).

- For the goals identified as potential use cases, what are the key pieces?
- For each outline level, what are key data?
- Outline all use cases.
- Prioritize the use-case flows.
- Decide on a final use-case list (for initial pass).

Step 3: Write a brief description of the use case(s).

- What two or three sentences describe all actors and the basic flow?
- Generate content first, and worry about wordsmithing it later.

Step 4: Detail the basic flow.

- What event starts the use case?
- How does the use case end?
- How does the use case repeat some behavior?
- What is the "happy" (best case) path?
- There is one and only one basic flow.

Step Wise Refinement of Use Case Model

Step 5: Detail the alternate flows.

- Are there optional situations for the use case?
- What might go wrong?
- What might not happen?
- Which resources might be blocked?
- Which alternate flows are special — perhaps nonfunctional — requirements (i.e., they apply to this use case only)?

Step 6: Review the use case(s).

- Are there more use cases?
- Should some use cases be redefined?
- Which ones can be combined?

Step 7: Record pre- and post-conditions.

- What was the previous state before this use case comes into play?
- What happens once the use case is complete?

Step 8: Develop generalizations for all use cases.

- Determine shared content and process for the use cases.
- What items have been noted for the glossary or as global business rules?
- Who has the most recent and accurate source document?
- Where is it located?

The Writing Process

1. Name the system scope and boundaries.

   Track changes to this initial context diagram with the in/out list.

2. Brainstorm and list the primary actors.

   Find every human and non-human primary actor, over the life of the system.

3. Brainstorm and exhaustively list user goals for the system.

   The initial Actor-Goal List is now available.

4. Capture the outermost summary use cases to see who really cares.

   Check for an outermost use case for each primary actor.

5. Reconsider and revise the summary use cases. Add, subtract, or merge goals.

   Double-check for time-based triggers and other events at the system boundary.

6. Select one use case to expand.

   Consider writing a narrative to learn the material.

Continued…

7. Capture stakeholders and interests, preconditions and guarantees.
   The system will ensure the preconditions and guarantee the interests.

8. Write the main success scenario (MSS).
   Use 3 to 9 steps to meet all interests and guarantees.

9. Brainstorm and exhaustively list the extension conditions.
   Include all that the system can detect and must handle.

10. Write the extension-handling steps.
    Each will end back in the MSS, at a separate success exit, or in failure.

11. Extract complex flows to sub use cases; merge trivial sub use cases.
    Extracting a sub use case is easy, but it adds cost to the project.

12. Readjust the set: add, subtract, merge, as needed.
    Check for readability, completeness, and meeting stakeholders' interests.

Exercise

Create a payroll system for restaurants and hotels. Make certain that the usual deductions are taken into consideration.

The payroll must accommodate both salaried and hourly employees.

The waiters are salaried employees, but the bus boys are hourly employees.

The payroll system must print checks weekly.

The system will produce a payroll register which will be turned over to auditors monthly.

Income and tax reports must be prepared according to legal requirements.

Reports concerning voluntary deductions will be prepared for various agencies on a quarterly basis.

The payroll must treat part-time employees as hourly employees.

Full-time employees and salaried employees may take advantage of the various company benefits, part-time employees may not.

Restaurant employees will be able to eat meals at their restaurant but will have the cost of the meals deducted from their paycheck.

Hotel employees will have room costs deducted if they live in the hotel.

There are voluntary deductions and mandatory government deductions that must be taken into account.

Exercise

- Identify for the Payroll System
  - Actors
  - Use cases

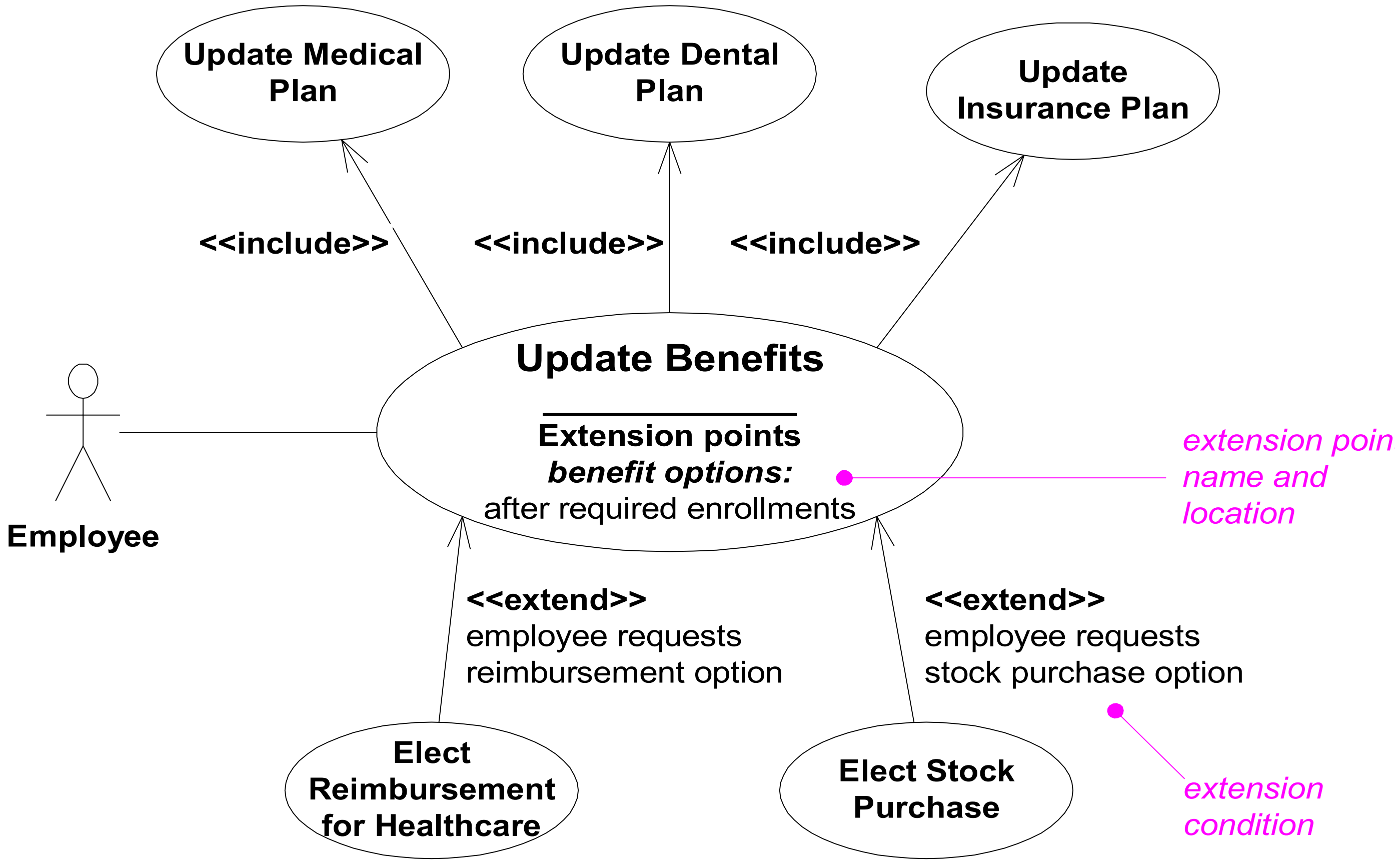
- Name the use cases

- Produce the use-case model for the Payroll System.

- Provide: Actor descriptions
  Brief use-case descriptions

- Compare your results to other groups.
  - Are there differences? Why? How would you resolve these differences?

Online HR System: Use Case Relationships


Update Medical Plan
Update Dental Plan
Update Insurance Plan
<<include>>
<<include>>
<<include>>
Update Benefits
Extension points
benefit options:
after required enrollments
Employee
extension point name and location
<<extend>>
employee requests reimbursement option
<<extend>>
employee requests stock purchase option
Elect Reimbursement for Healthcare
Elect Stock Purchase
extension condition

Online HR System: Update Benefits Use Case

- Actors: employee, employee account db, healthcare plan system, insurance plan system

- Preconditions:
  - Employee has logged on to the system and selected "update benefits" option

- Basic course
  - System retrieves employee account from employee account db
  - System asks employee to select medical plan type; include Update Medical Plan.
  - System asks employee to select dental plan type; include Update Dental Plan.
  - ...

- Alternative courses
  - If health plan is not available in the employee’s area the employee is informed and asked to select another plan...

When to model use cases

- Model user requirements with use cases.
- Model test scenarios with use cases
- If you are using a use-case driven method
  - start with use cases and derive your structural and behavioral models from it.
- If you are not using a use-case driven method
  - make sure that your use cases are consistent with your structural and behavioral models.

Bank ATM Example

- An automated teller machine (ATM) or the automatic banking machine (ABM) is banking subsystem (subject) that provides bank customers with access to financial transactions in a public space without the need for a cashier, clerk or bank teller.

- Customer (actor) uses bank ATM to check balances of his/her bank accounts

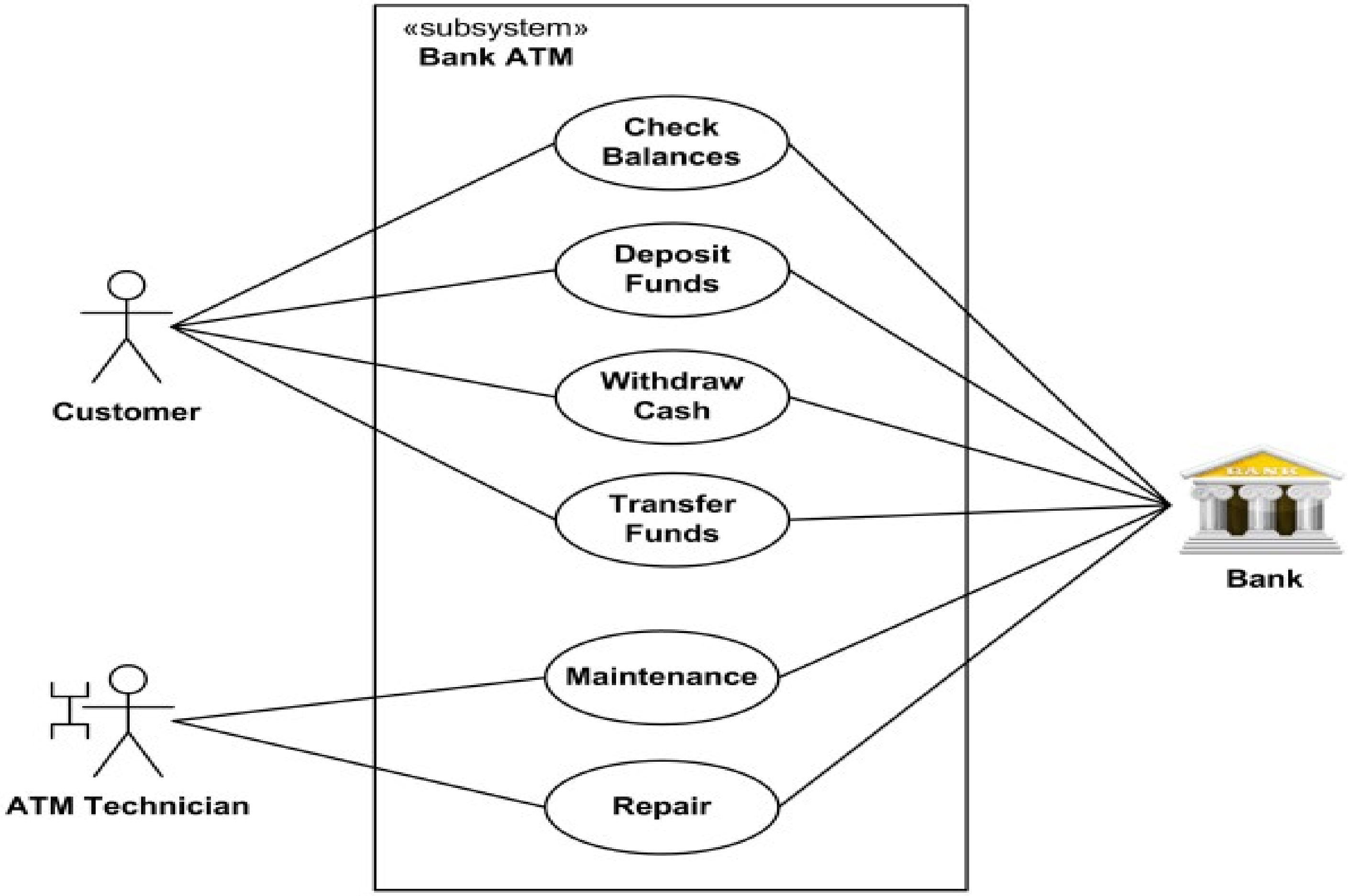
- Deposit funds, withdraw cash and/or transfer funds (use cases).

Bank ATM Example

- ATM Technician provides maintenance and repairs.

- All these use cases also involve Bank actor whether it is related to customer transactions or to the ATM servicing.

Use Case Diagram


«subsystem»
Bank ATM
Check Balances
Deposit Funds
Withdraw Cash
Transfer Funds
Maintenance
Repair
Customer
ATM Technician
BANK
Bank

Use Case Diagram

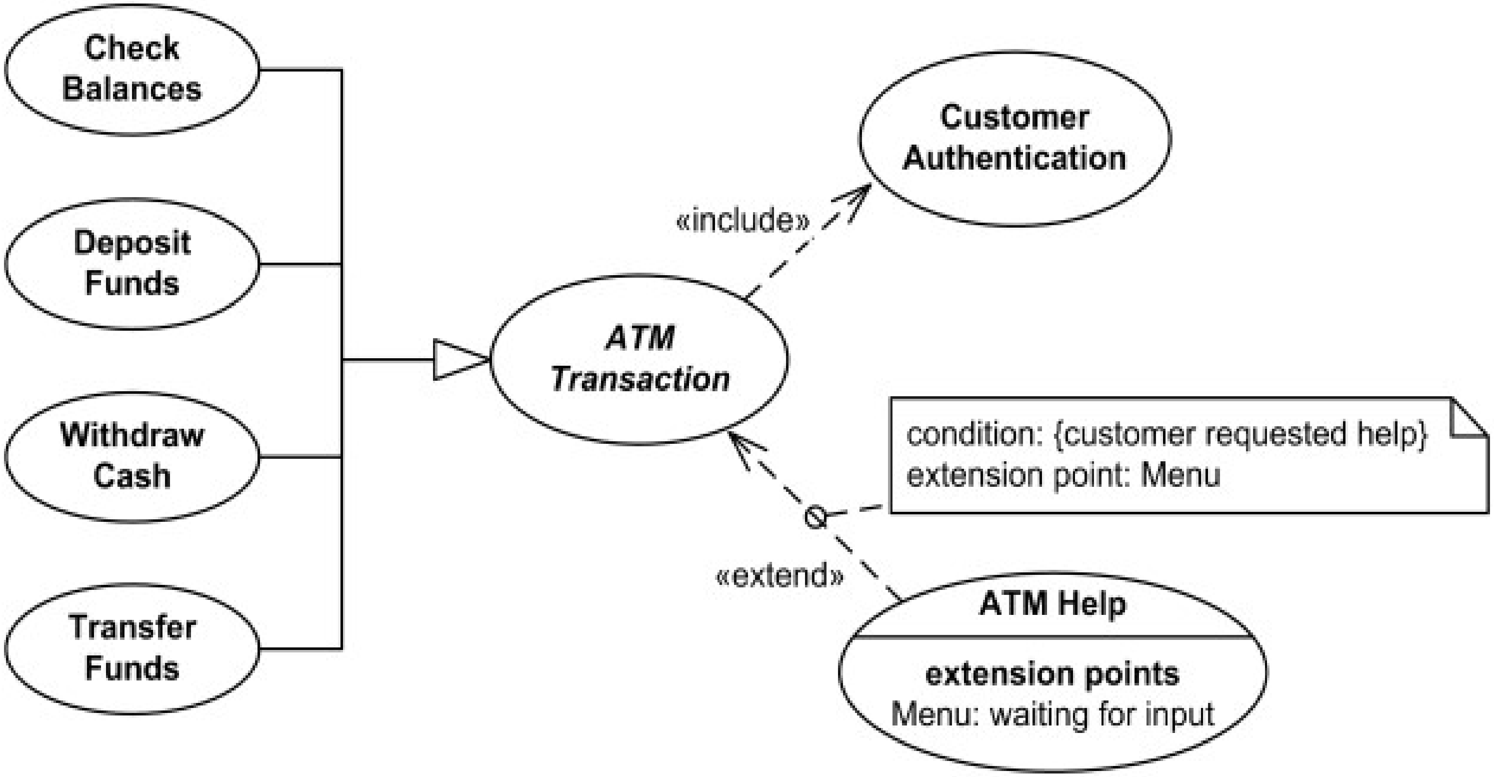
- On most bank ATMs, the customer is authenticated by inserting a plastic ATM card and entering a personal identification number (PIN).

- **Customer Authentication** use case is **required** for every ATM transaction so we show it as include relationship.

- Including this use case as well as transaction generalizations make the **ATM Transaction** an abstract use case

- Bank ATM Transactions and Customer Authentication Use Cases Example.
- If needed, customer may ask ATM for **help.ATM Transaction** use case is extended via **Menu** extension point by the **ATM Help** use case whenever **ATM Transaction** is at the location specified by the **Menu** and the bank customer requests help, e.g. by selecting Help menu item.

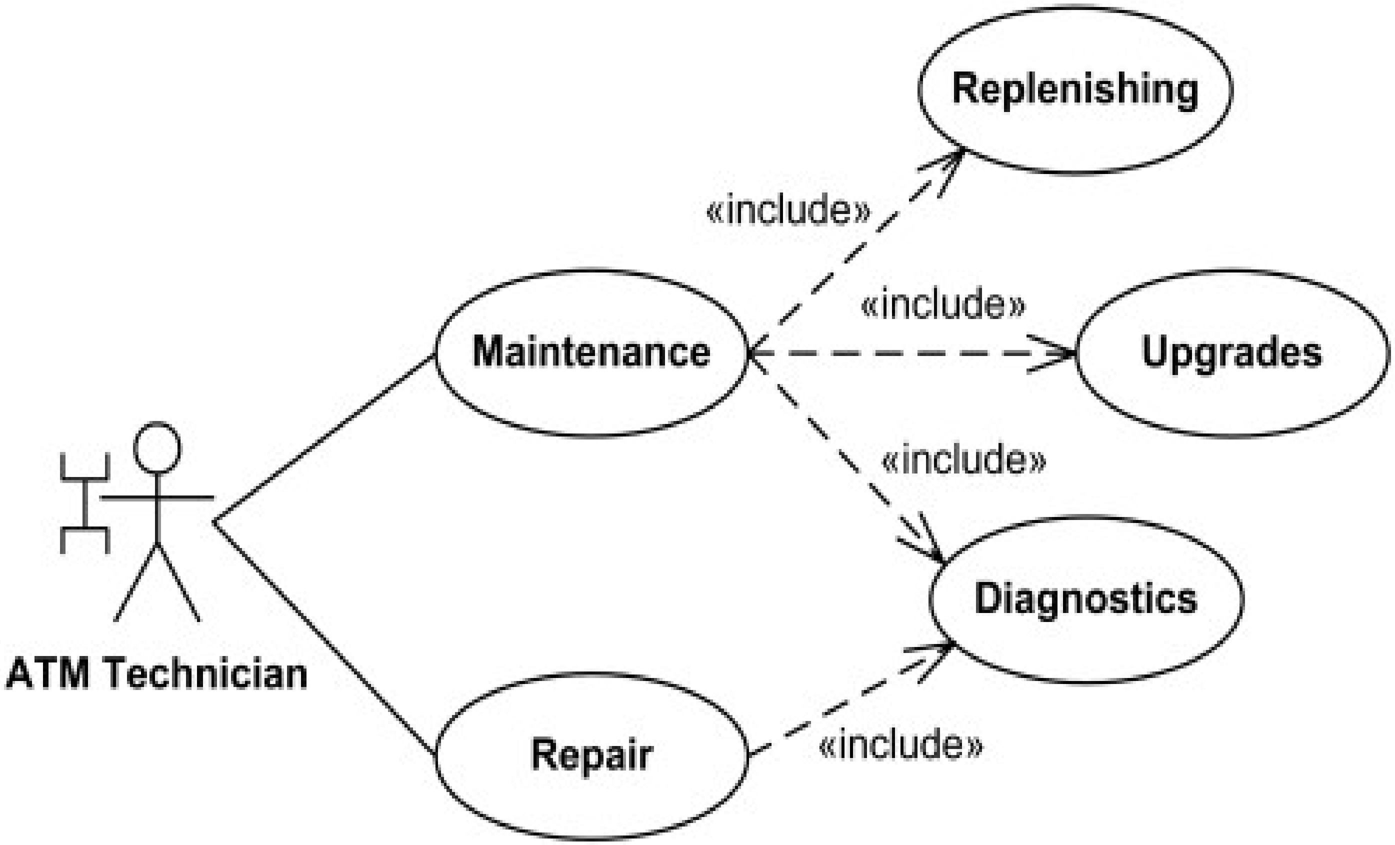
Use Case Diagram


Check Balances
Deposit Funds
Withdraw Cash
Transfer Funds
ATM Transaction
«include»
Customer Authentication
condition: {customer requested help}
extension point: Menu
«extend»
ATM Help
extension points
Menu: waiting for input

Use Case Diagram

- Bank ATM Maintenance, Repair, Diagnostics Use Cases Example.
- **ATM Technician** maintains or repairs Bank ATM.
- **Maintenance** use case includes **Replenishing** ATM with cash, ink or printer paper,
- **Upgrades** of hardware, firmware or software, and remote or on-site **Diagnostics**.
- **Diagnostics** is also included in (shared with) **Repair** use case.

Use Case Diagram


Replenishing
«include»
Maintenance
«include»
Upgrades
«include»
ATM Technician
Diagnostics
Repair
«include»

Business use case diagram for Restaurant - External view

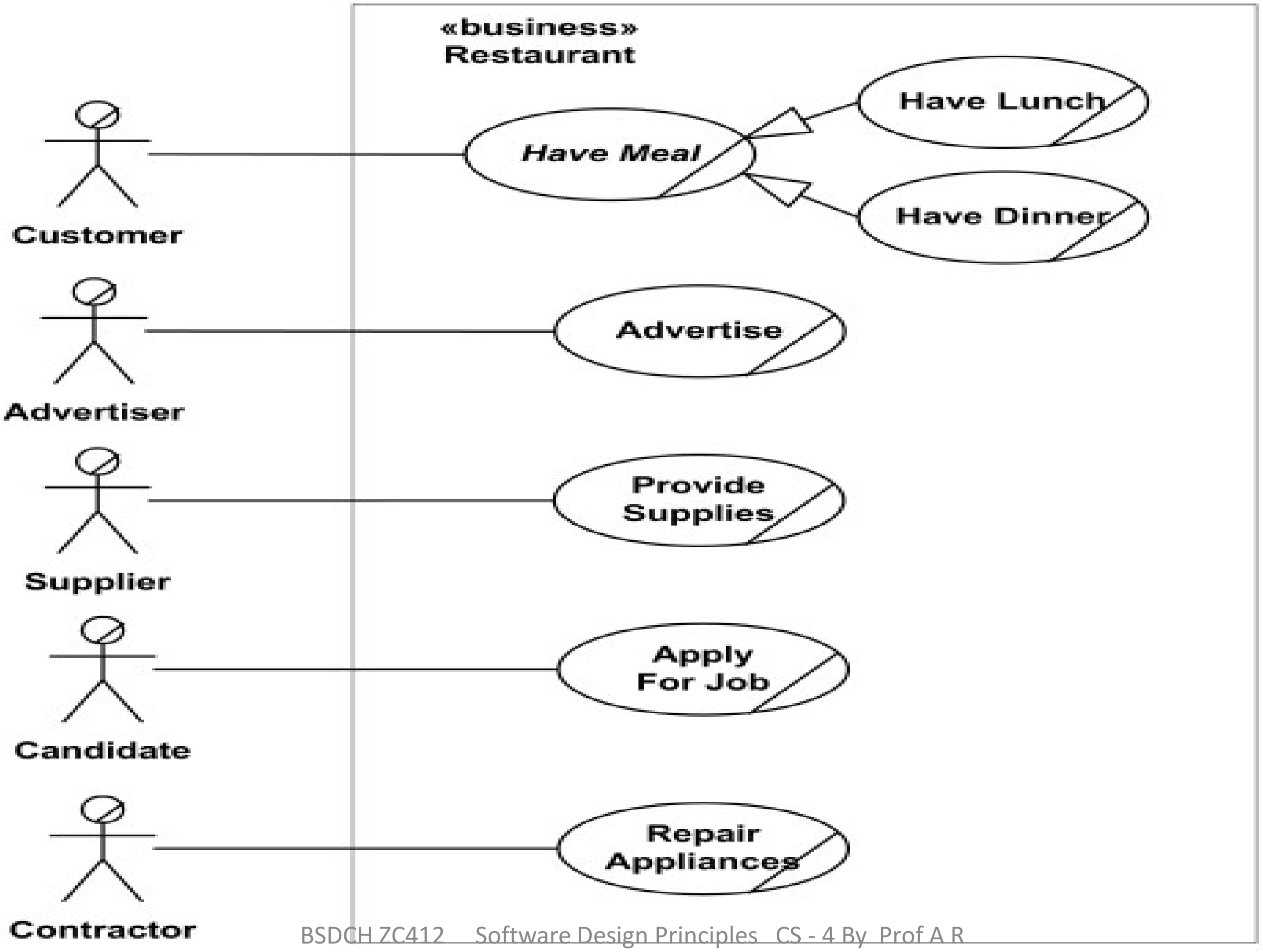
- Customer wants to Have Meal, Candidate - to Apply for Job, and Contractor - to fix some appliances.

- Note, that we don't have such actors as Chef or Waiter.

- They are not external roles but part of the business we model - the Restaurant, thus - they are not actors.

- In terms of RUP Chef and Waiter are business workers.

Continues..


«business»
Restaurant
Customer
Advertiser
Supplier
Candidate
Contractor
Have Meal
Have Lunch
Have Dinner
Advertise
Provide Supplies
Apply For Job
Repair Appliances

Business use case diagram for Restaurant - Internal view

- This shows **internal business view** of a restaurant.

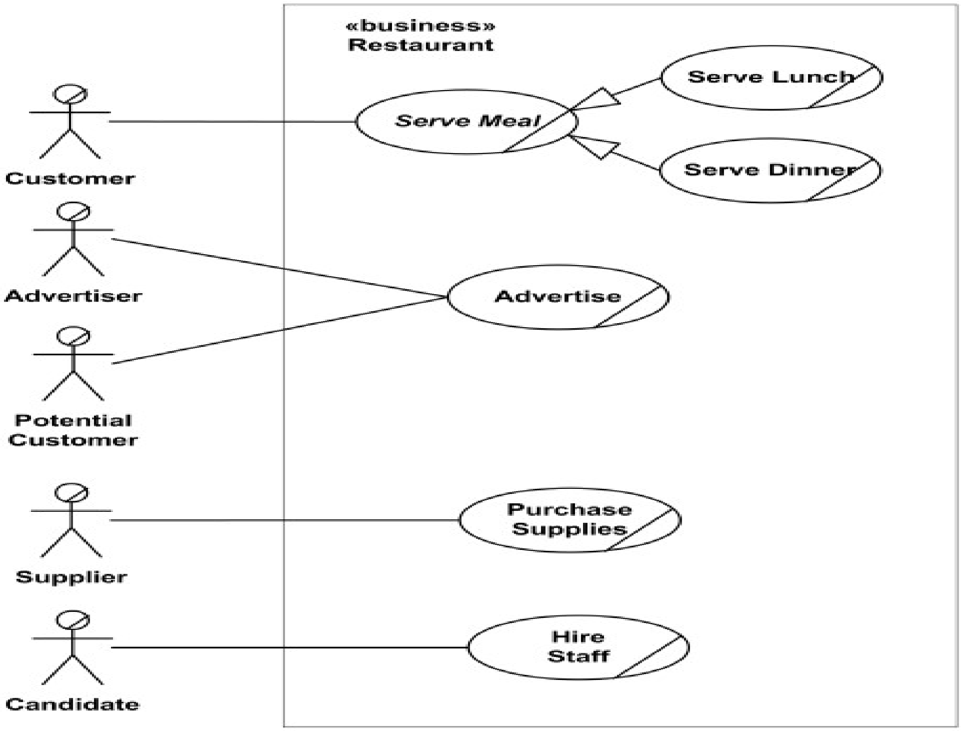
- In this case we can see that restaurant has several business processes represented by business use cases which provide some services to external business actors.

- **Customer** is now connected to **Serve Meal** use case, **Supplier** - to **Purchase Supplies**.

- We have now new actor **Potential Customer** participating in **Advertise** use case by reading ads and getting some information about restaurant.

Continues..

- At the same time, Contractor actor is gone because Repair Appliances is not a service usually provided by restaurants.


«business»
Restaurant
Customer
Serve Meal
Serve Lunch
Serve Dinner
Advertiser
Advertise
Potential Customer
Supplier
Purchase Supplies
Candidate
Hire Staff

Analysis

Analysis models

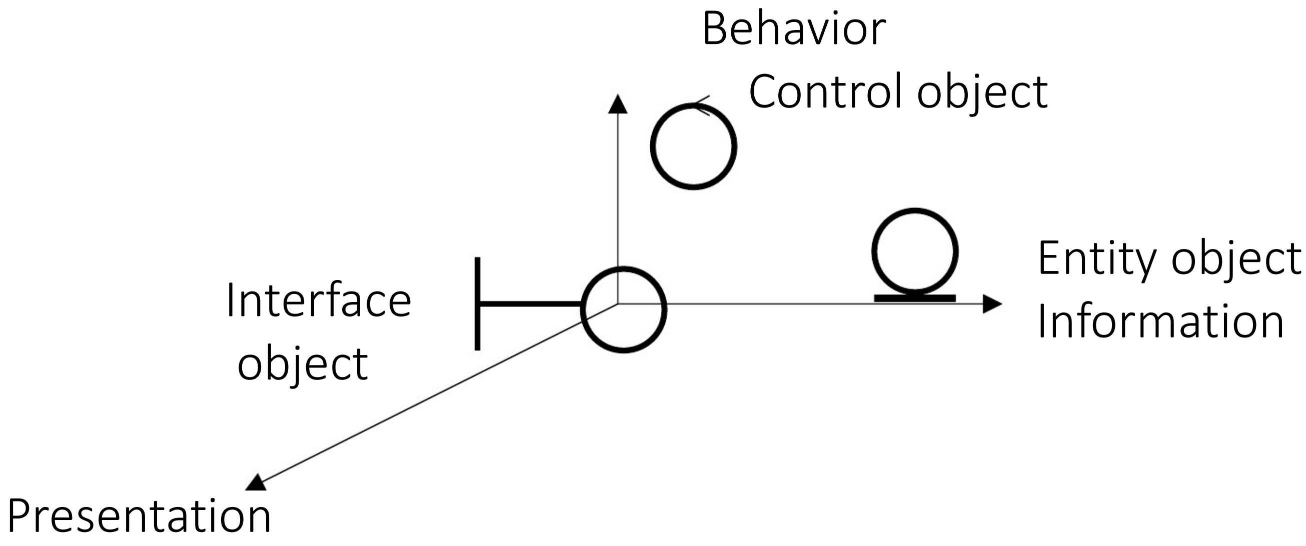
Language of the developers Capture the structure of the system with the use cases

Structured by classes

Outlines how to realize functionality

Analyze Use Case

Stable, Robust & Maintainable structure that is Extensible Capture information, behavior & presentation


Behavior
Control object
Interface object
Entity object
Information
Presentation

Object Types in Analysis Model

Entity Object :

models information & behavior coupled with the information

Interface Object :

models behavior & information dependent on interface to the system

Control Object :

models behavior of operating on several different entity objects

Provides stable system where change localized

Boundary classes

Interaction between actor and system Fairly higher-level Windows, forms, printer, sensor, APIs

Related to at least one actor and vice-versa

Association & Aggregation of Boundary Objects

Boundary objects may communicate with other boundary objects

Boundary objects may contain other boundary objects

Use-Case Analysis

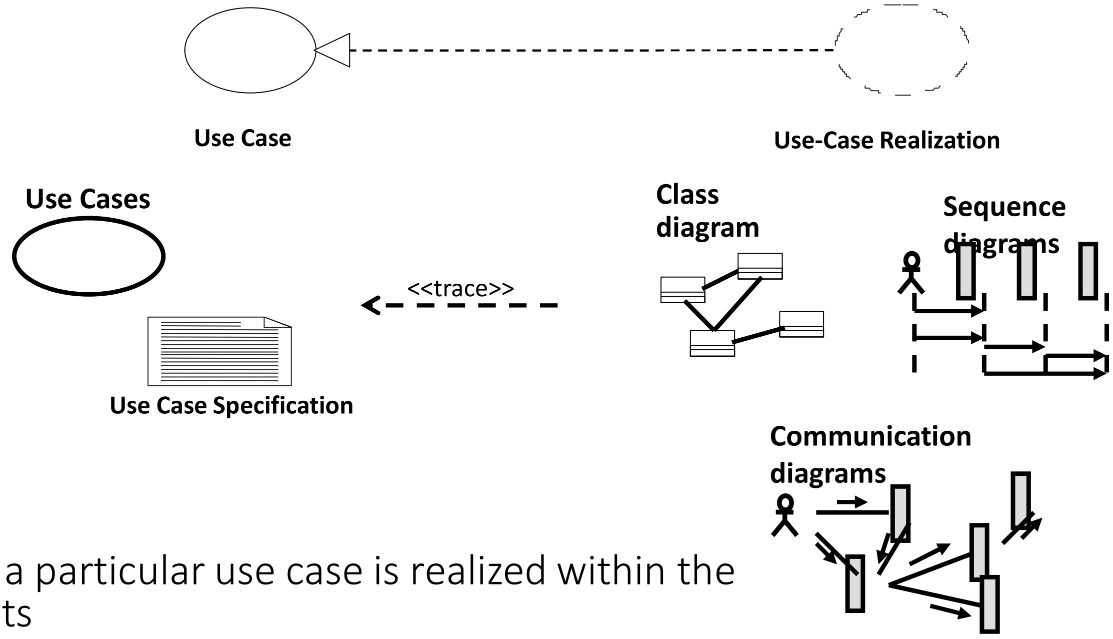
Analyze the requirements to discover what the system objects are

These slides capture the use-case analysis activity of the Rational Unified Process

Use-Case Analysis

- Use-case analysis is where the requirements meet object-orientation
  - Recall: in the Unified Process, the use-case model is the primary artifact in the requirements model
  - In use-case analysis, identify the classes which perform a use-case flow of events
  - Distribute the use-case behavior to those classes
    - Identifying the responsibility of the classes
  - Develop use case realizations that model the collaborations between instances of the identified classes
    - How the class instances work together to deliver the requirements
- The result is a first-draft, rough-cut of the system object model
  - An abstraction of the design model; refined during design

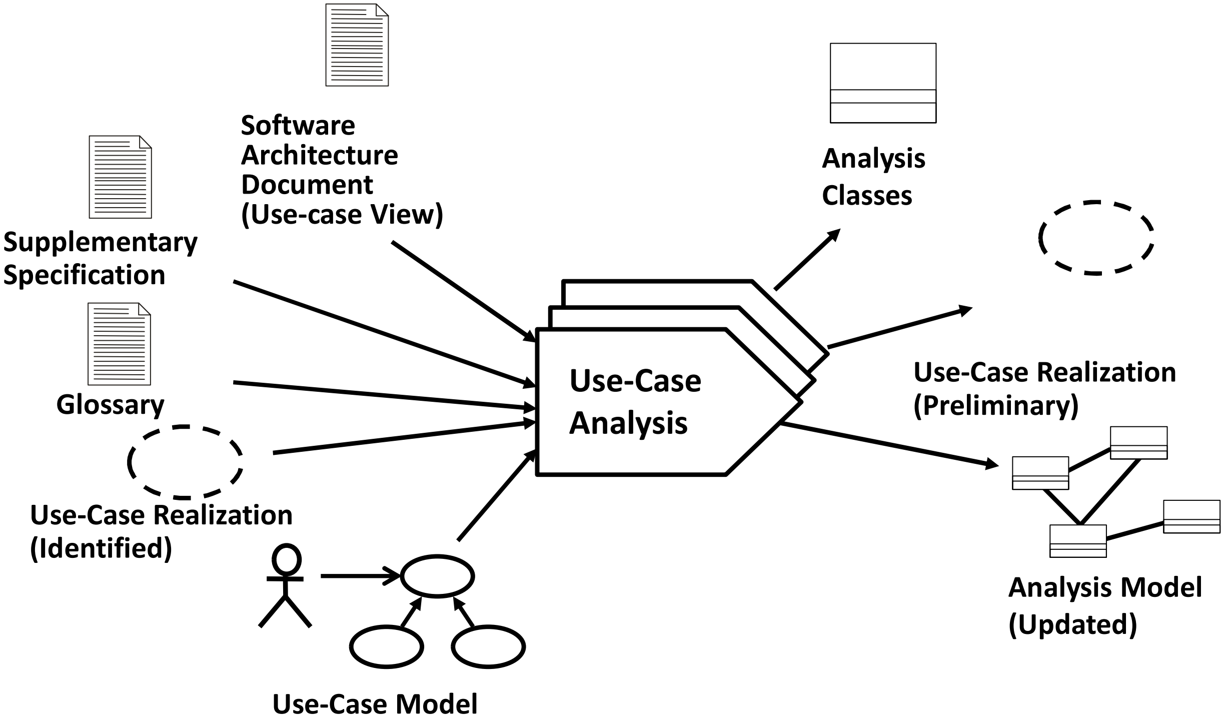
Review: Use-Case Realization


Use Case
Use-Case Realization
Use Cases
Use Case Specification
<<trace>>
Class diagram
Sequence diagrams
Communication diagrams

- A use-case realization is a description of how a particular use case is realized within the design model, in terms of collaborating objects

- It is one possible realization, corresponding to a specific selection among design options

Use-Case Analysis


Software
Architecture
Document
(Use-case View)
Supplementary
Specification
Glossary
Use-Case Realization
(Identified)
Use-Case Model
Use-Case
Analysis
Analysis
Classes
Use-Case Realization
(Preliminary)
Analysis Model
(Updated)

Instantiate the activity once per use case

Use-Case Analysis - Steps

- Supplement the Use-Case Description
- For each use-case realization
  - Find classes from use-case behavior
  - Distribute use-case behavior to classes
- For each resulting analysis class
  - Describe responsibilities
  - Describe attributes and associations
  - Qualify architectural analysis mechanisms
- Unify analysis classes
- Checkpoints

Use-Case Analysis - Steps

Next

- Supplement the Use-Case Description

- For each use-case realization

  - Find classes from use-case behavior
  - Distribute use-case behavior to classes

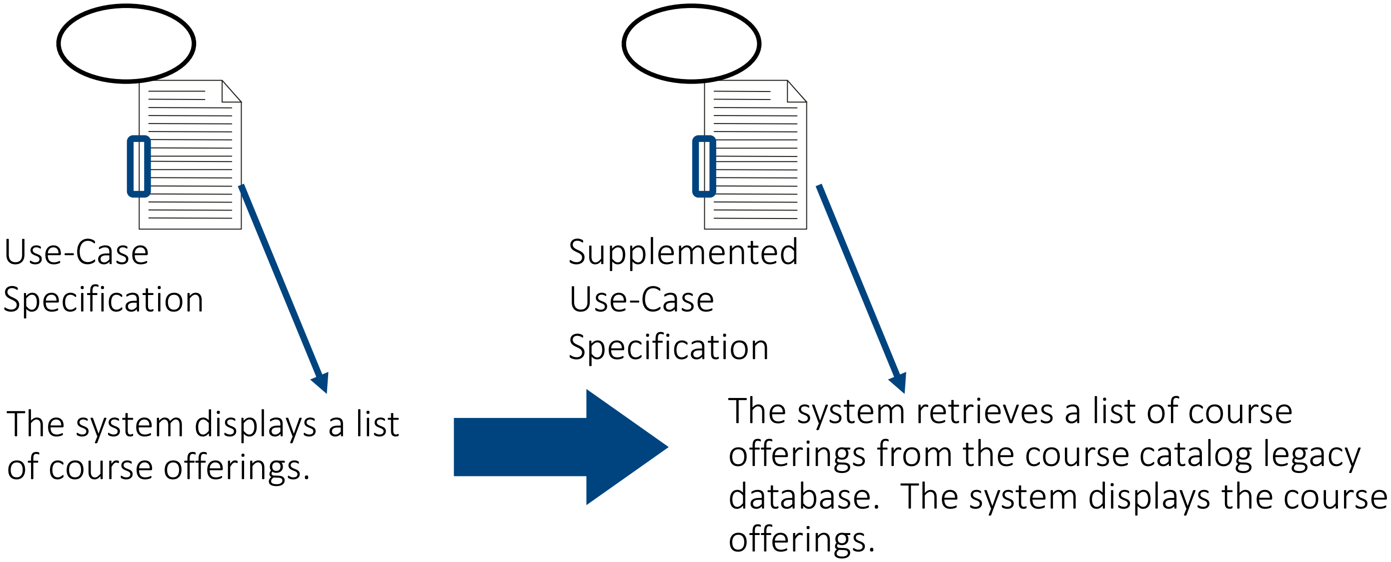
- For each resulting analysis class

  - Describe responsibilities
  - Describe attributes and associations
  - Qualify architectural analysis mechanisms

- Unify analysis classes

- Checkpoints

Supplement the Use-Case Description



Use-Case Specification

The system displays a list of course offerings.

Supplemented Use-Case Specification

The system retrieves a list of course offerings from the course catalog legacy database. The system displays the course offerings.

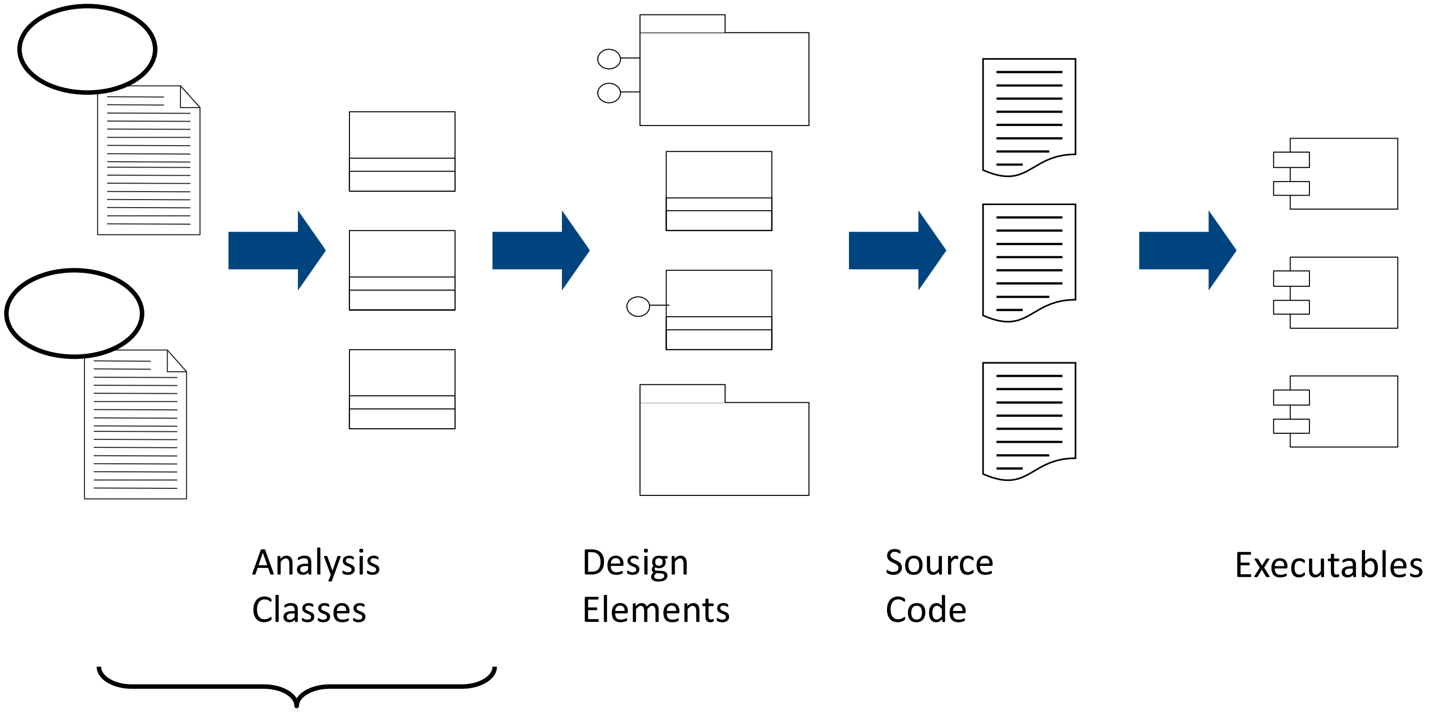
- Capture additional information needed in order to understand the required internal behavior of the system that may be missing from the use-case description written for the customer of the system.
- Note: exposes some solution structure – not appropriate for requirements, but necessary to define objects.

Use-Case Analysis - Steps

- Supplement the Use-Case Description
- For each use-case realization
    - Find classes from use-case behavior
    - Distribute use-case behavior to classes
- For each resulting analysis class
    - Describe responsibilities
    - Describe attributes and associations
    - Qualify architectural analysis mechanisms
- Unify analysis classes
- Checkpoints

Next

Analysis Classes: A First Step Towards Executables



Analysis
Classes

Design
Elements

Source
Code

Executables

Use-Case Analysis

The Analysis Model is Temporary

- The analysis model is an early conceptual model of how the system will work

  - It evolves quickly
  - It is fluid, changing as different representations and their implications are explored

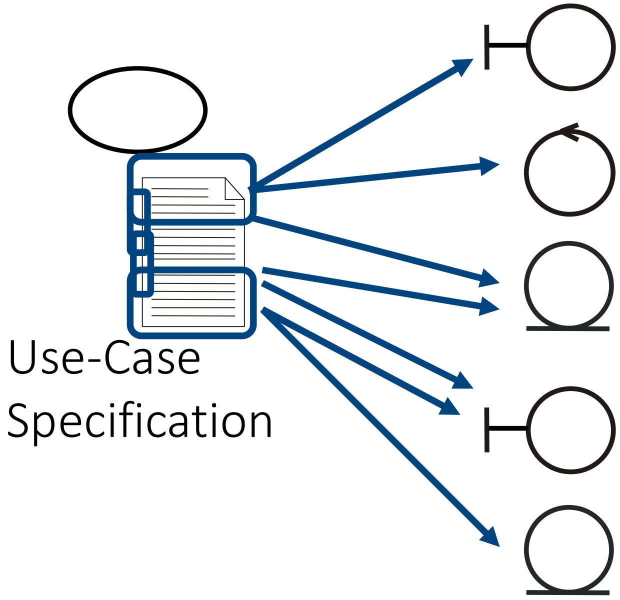
- Analysis classes rarely survive into design unchanged

  - A given analysis class often represents collaborations of multiple design objects, encapsulated by subsystems
  - Think of analysis classes as “proto-classes” representing “clumps of behavior”
    - Allow us to explore alternative distribution of responsibilities to achieve a good separation of concerns

- Be careful to not spend too much time on “formal” documentation and a well-polished model
  - Consider the analysis model as informal
  - “Structured doodling”, artist “studies”

Find Classes from Use-Case Behavior

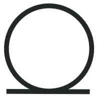
- The complete behavior of a use-case must be allocated to analysis classes


Use-Case
Specification

Class Stereotypes for Analysis

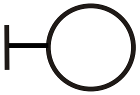
- Distinguish and separate the concerns of system interface from application logic/control flow from persistent application objects

- Specialize the class object in UML to represent the distinctions



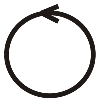
Entity

  - Entity classes: system information
    - Long-lived, real-life object or event in the application domain
    - Data and behavior
    - Usually persistent (saved in a file or database)



Boundary

  - Boundary classes: system boundary
    - Interaction between the system and its actors
    - At the system boundary



Control

  - Control classes: use-case behavior coordination
    - Coordination and sequencing of system behavior
    - Transactions

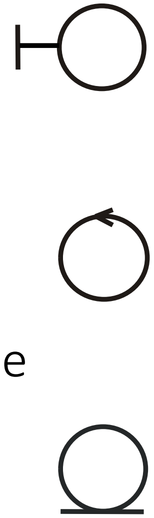
Analysis Classes

- The distinction into the three analysis class types helps
  - Clarify and separate class roles in the system
  - Separate things that tend to change separately

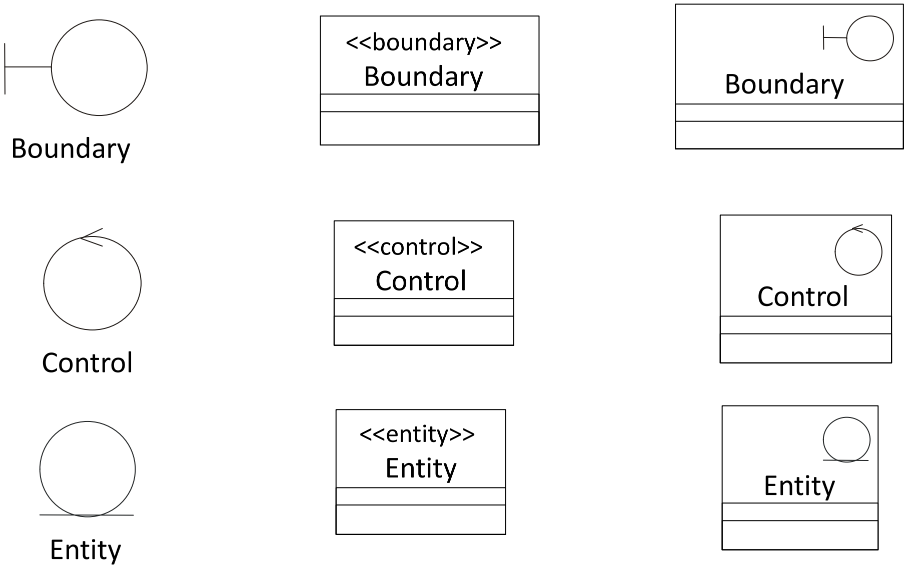
- Classes identified in analysis need not have a stereotype

- Once the roles have been analyzed and there is a good system decomposition, the distinction between types is no longer really useful

  - The distinction (and the stereotypes) tend to go away in design



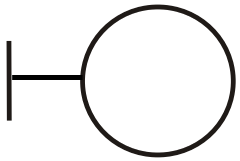
Alternate Visualizations of the Same Thing


Boundary
<<boundary>>
Boundary
Boundary
Control
<<control>>
Control
Control
Entity
<<entity>>
Entity
Entity

- UML allows multiple graphical representations (“syntax”) of the same conceptual (semantic) model element

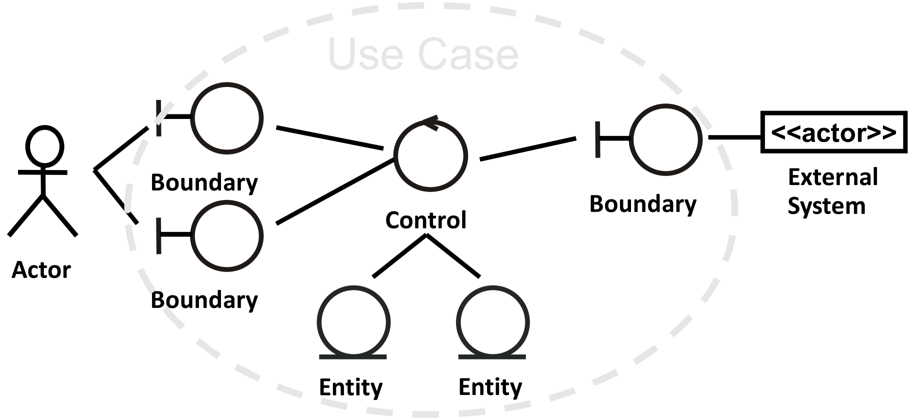
Boundary Classes

- A boundary class intermediates between the system and something outside the system
  - Insulate the system from changes in the surroundings (changes in external systems, user requirements, etc.)



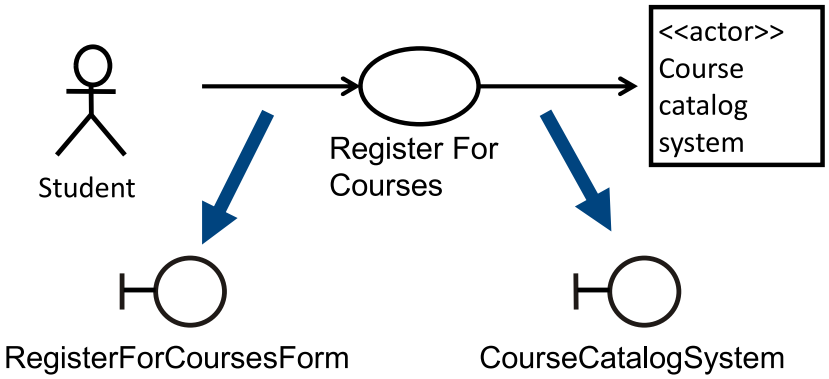
- Types
  - User interface classes
    - Display windows/screens, keyboard, microphone voice recognition, motion tracker, etc.
  - System interface classes
    - Interface to external systems, legacy systems, use of an external Application Programming Interface (API)
  - Device interface classes
    - Interface to devices which detect external events
    - Capture the responsibilities of a device or sensor

The Role of Boundary Classes


Use Case
Boundary
Boundary
Actor
Control
Boundary
<<actor>>
External System
Entity
Entity

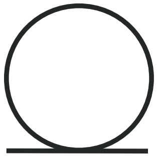
- Model interaction between the system’s surroundings and its inner workings
  - Transform and translate events
  - Note changes in the system presentation (such as a user interface display)
- Make it easier to understand and clarify system boundaries
- Help identify related services
  - For example, identifying a printer interface suggests the need to format printouts
- Insulate external forces from system inner workings and vice-versa
  - For example, changing a communication protocol or GUI look-and-feel should mean only changing the boundary classes, not the entity and control classes

Finding Boundary Classes


Student
Register For Courses
<<actor>> Course catalog system
RegisterForCoursesForm
CourseCatalogSystem

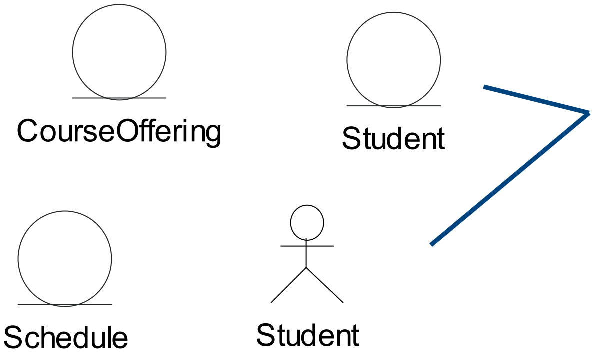
- Guideline: Start with one boundary class per actor/use-case pair
- Concentrate on the responsibilities (NOT the details)
  - User interface classes
    - Concentrate on what information is presented (NOT on the UI details of graphics, layout, style, controls, etc.)
  - System and device interface classes
    - Concentrate on what protocols must be defined (NOT on how the protocols will be implemented) – responsibilities, not details
    - If there is already a working communication with the external system or device, make note of it for later reference during design
- Consider the source of all external events and make sure there is a way for the system to detect these events

Entity Classes



- Entity objects represent the key concepts of the system being developed
  - Store and manage information in the system
    - Data (usually persistent)
    - Structure (usually persistent)
    - Behavior
- Usually not specific to one use-case
- Examples
  - Banking: *Account*, *Customer*
  - Network Management: *Node, Link*
  - Information the system needs to know about its actors
    - A surrogate for the actor – the actor itself is separate
- Identify the entity classes by name and brief description

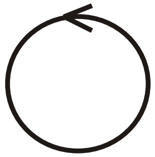
Finding Entity Classes



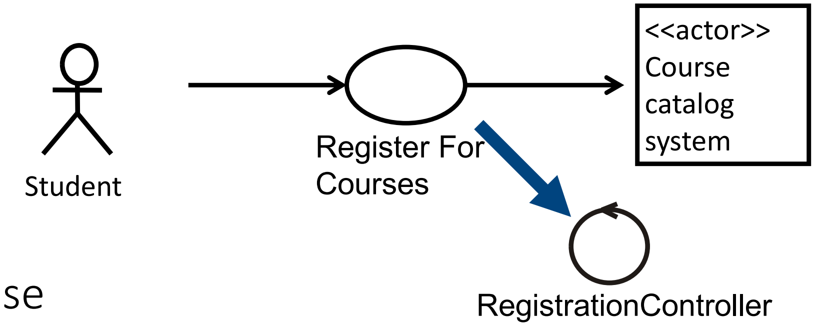
Different!
Consider “StudentInfo” or “StudentActor” as names to clarify the distinction

- Sources
  - Glossary, business/domain model, use-case flow of events, key abstractions (from architectural analysis)

- A technique: study the nouns
  - Underline noun clauses in the use-case flow of events
  - Remove redundant candidates
  - Remove vague candidates (or make them clear)
  - Remove actors (out of scope, but they may have surrogate entity classes)
  - Remove implementation constructs
  - Remove attributes (save for later)
  - Remove operations



Control Classes



Student

Register For Courses

<<actor>>
Course
catalog
system

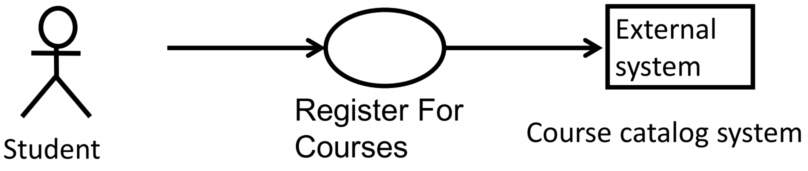
RegistrationController

- Use-case behavior coordinator
    - Typically, one control class per use case
        - Create control object at start of use-case, delete it at end
    - Delegates actor-visible behavior to the entity classes
        - “Orchestrate and Delegate Behavior”
    - If the use-case is simply accessing and changing information, a control class may be unnecessary (boundary classes and entity classes interact directly)

- Examples
    - Transaction management
    - Resource coordination
    - Error handling
    - Decision logic (control the flow of events; state-dependent flows)

- May disappear in design – become methods on UI classes
    - “On MouseEvent do X”

Analysis Classes for “Register for Courses”



Student

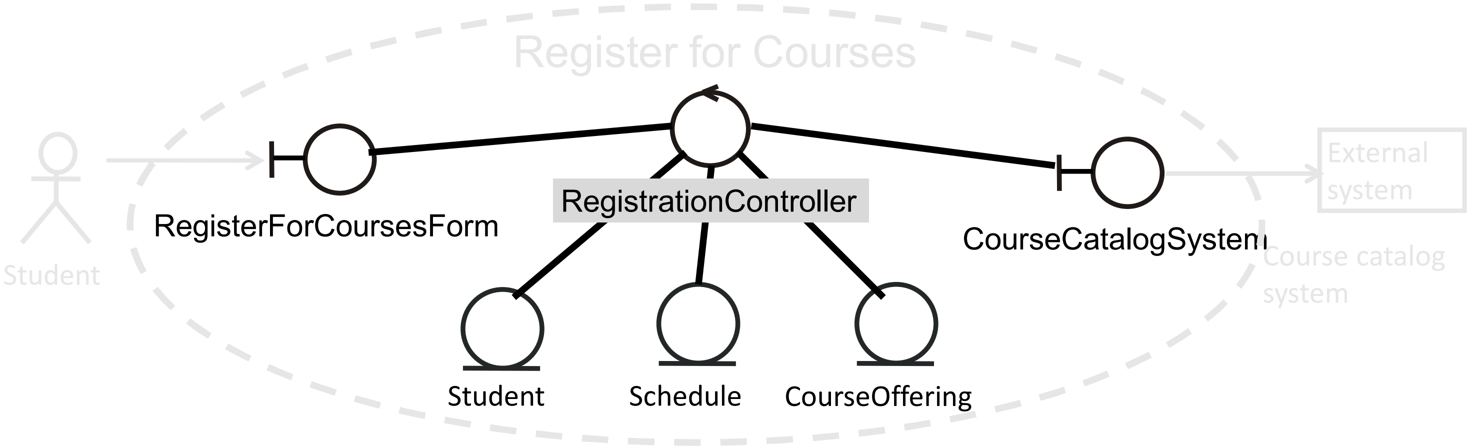
Register For Courses

External system

Course catalog system

Requirements (Use-Case) Model

Analysis Model – View of Participating Classes in Use-Case Realization



Register for Courses

Student

RegisterForCoursesForm

RegistrationController

CourseCatalogSystem

External system

Course catalog system

Student

Schedule

CourseOffering

Use-Case Analysis - Steps

- Supplement the Use-Case Description

Next

- For each use-case realization
  - Find classes from use-case behavior
  - Distribute use-case behavior to classes

- For each resulting analysis class
  - Describe responsibilities
  - Describe attributes and associations
  - Qualify architectural analysis mechanisms

- Unify analysis classes

- Checkpoints

Distribute Use-Case Behavior to Classes

- For each use-case flow of events
  - Identify participating analysis classes
  - Allocate use-case responsibilities to those analysis classes
  - Model analysis class interactions in interaction diagrams

    - One interaction diagram for each variant of a use-case’s flow of events
      - Flows for different user options
      - Exception-handling flows
      - etc.

    - In analysis, collaboration diagrams (as opposed to sequence diagrams) help focus on class responsibilities

      - Even so, I tend to use sequence diagrams to capture details and trace flow of control

Guidelines: Allocating Responsibilities to Classes

- Use analysis class stereotypes as a guide
    - Boundary classes
        - Behavior that involves communication with an actor
    - Entity classes
        - Behavior that involves the data encapsulated within the abstraction
    - Control classes
        - Behavior specific to a use case or part of a very important flow of events

(continued)

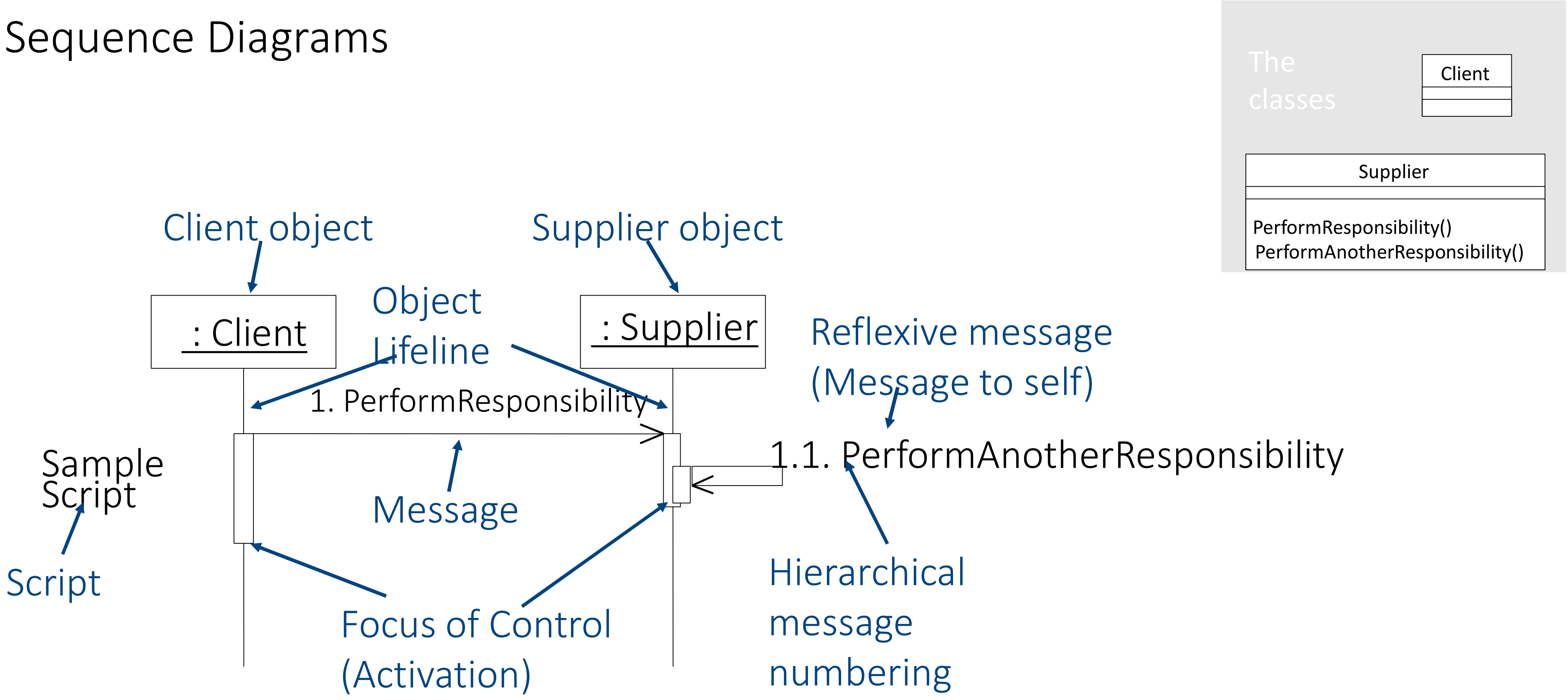
Guidelines: Allocating Responsibilities to Classes (continued)

- Who has the data needed to perform the responsibility?
  - If one class has the data, put the responsibility with the data
  - If multiple classes have the data
    - Put the responsibility with one class and add a relationship to the other
    - Create a new class, put the responsibility on the new class, and add relationships to classes needed to perform the responsibility
    - Put the responsibility on the control class, and add relationships to classes needed to perform the responsibility

- Keep clear the responsibility: one object's need to know information is not the same as an object's responsibility to provide information and manage information: client vs. supplier

- Refactor
  - Re-allocate data and responsibility among classes
  - Reuse: Assign new responsibility to a class that has similar responsibility

Sequence Diagrams



The classes

Client

Supplier

PerformResponsibility()
PerformAnotherResponsibility()

Client object

Supplier object

: Client

: Supplier

Object Lifeline

Reflexive message (Message to self)

1. PerformResponsibility

1.1. PerformAnotherResponsibility

Sample Script

Message

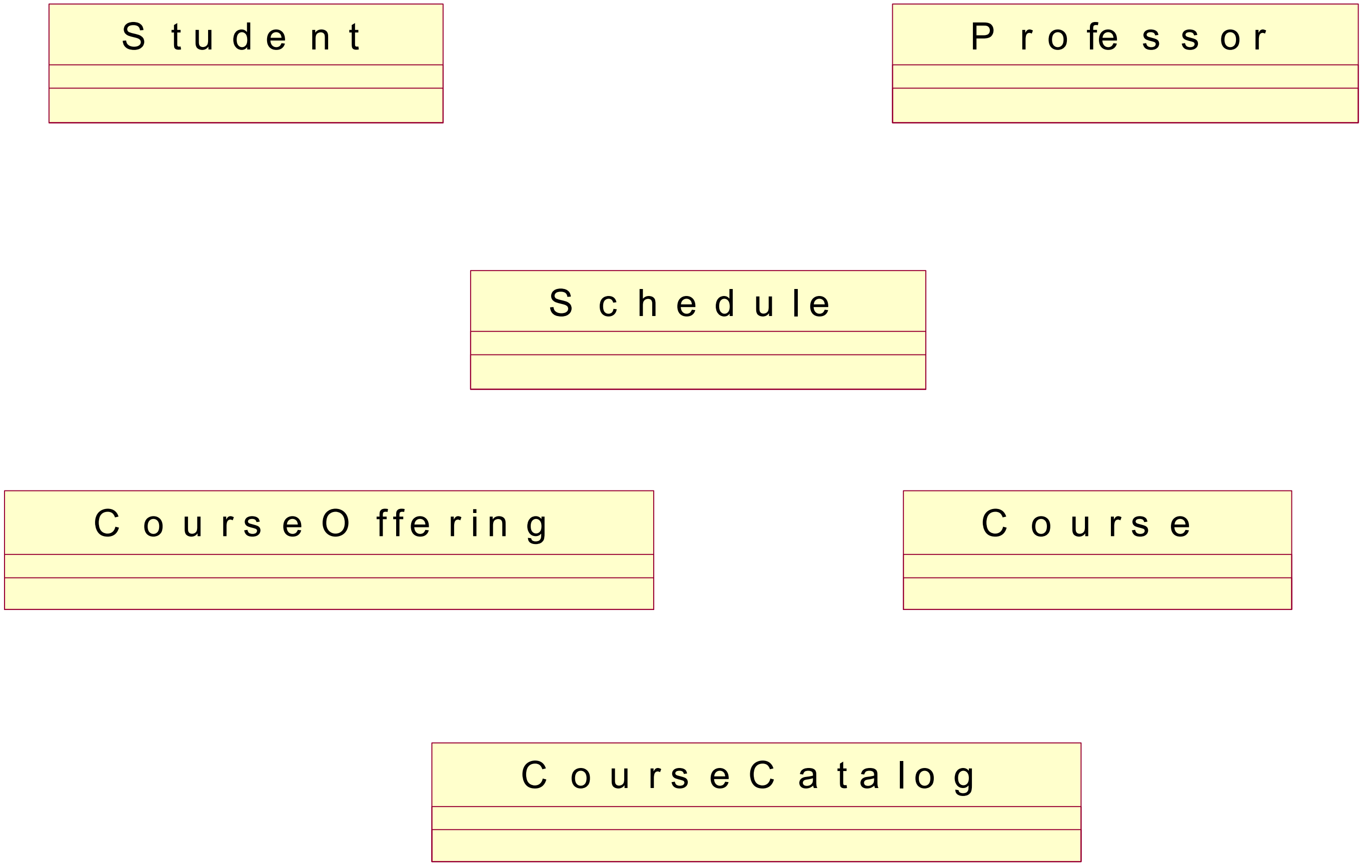
Script

Focus of Control (Activation)

Hierarchical message numbering

Example: College Course Registration System

Key Abstractions (from Architectural Analysis activity)



Student

Professor

Schedule

CourseOffering

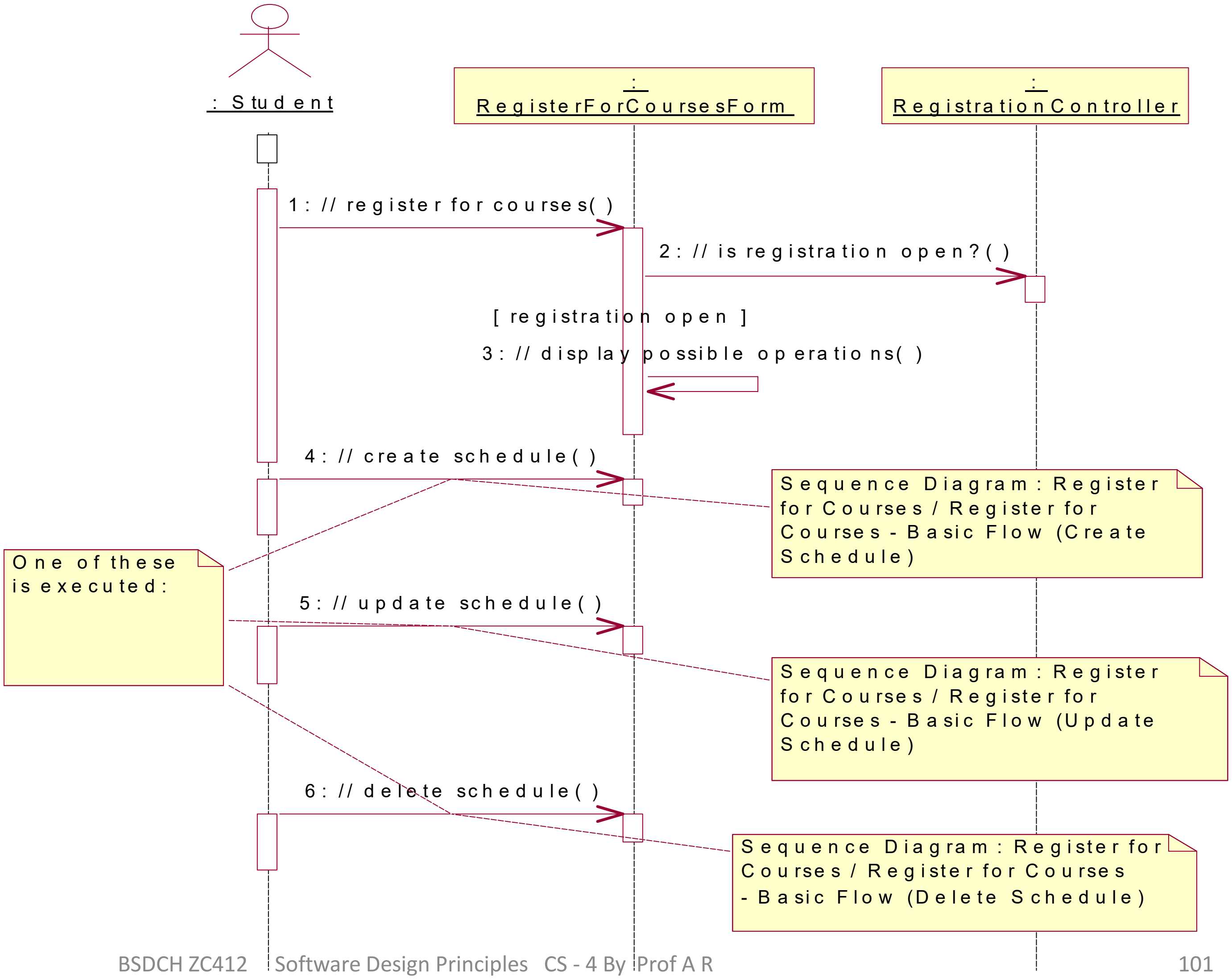
Course

CourseCatalog

Build a Sequence Diagram for "Register for Courses"

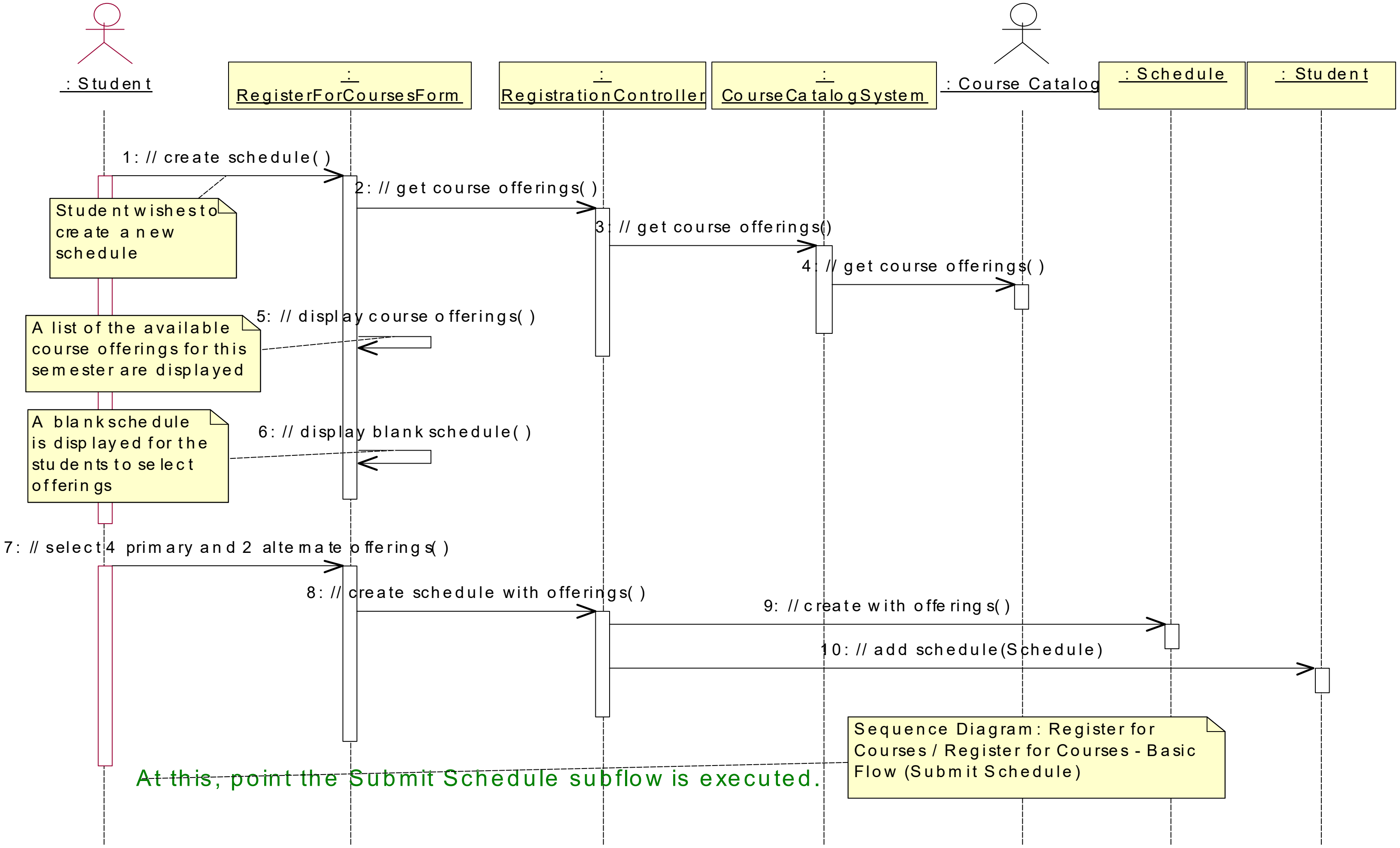
Basic flow, top level

(The basic flow has six diagrams capturing different portions of the flow)


: Student
: RegisterForCoursesForm
: RegistrationController
1 : // register for courses( )
2 : // is registration open?( )
[registration open]
3 : // display possible operations( )
4 : // create schedule( )
5 : // update schedule( )
6 : // delete schedule( )
One of these is executed:
Sequence Diagram : Register for Courses / Register for Courses - Basic Flow (Create Schedule)
Sequence Diagram : Register for Courses / Register for Courses - Basic Flow (Update Schedule)
Sequence Diagram : Register for Courses / Register for Courses - Basic Flow (Delete Schedule)

Build a Sequence Diagram for "Register for Courses"

Basic flow, create a schedule portion


: Student
: RegisterForCoursesForm
: RegistrationController
: CourseCatalogSystem
: Course Catalog
: Schedule
: Student
1: // create schedule( )
2: // get course offerings( )
3: // get course offerings( )
4: // get course offerings( )
5: // display course offerings( )
6: // display blank schedule( )
7: // select 4 primary and 2 alternate offerings( )
8: // create schedule with offerings( )
9: // create with offerings( )
10: // add schedule(Schedule)

Student wishes to create a new schedule

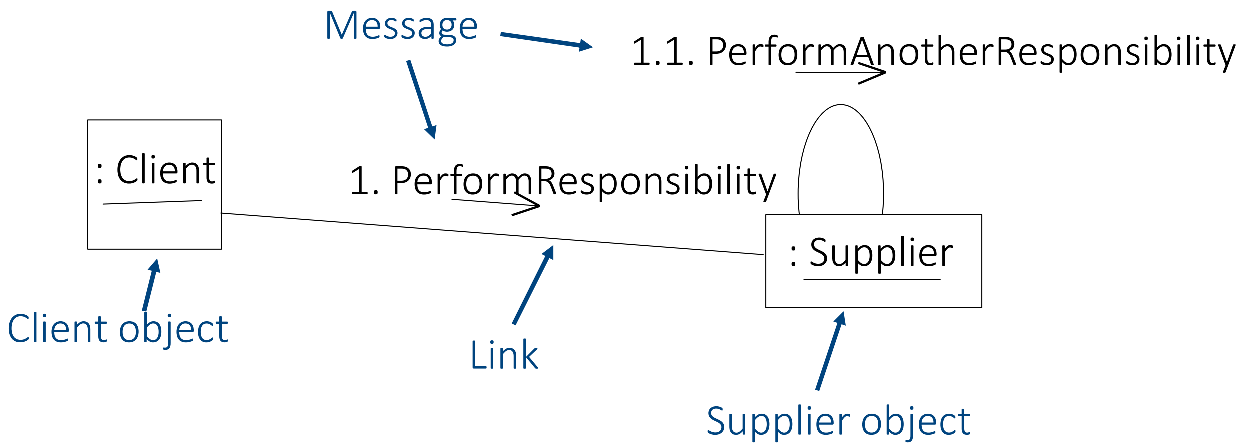
A list of the available course offerings for this semester are displayed

A blank schedule is displayed for the students to select offerings

At this, point the Submit Schedule subflow is executed.

Sequence Diagram : Register for Courses / Register for Courses - Basic Flow (Submit Schedule)

Collaboration Diagrams


Message
1.1. PerformAnotherResponsibility
: Client
1. PerformResponsibility
: Supplier
Client object
Link
Supplier object

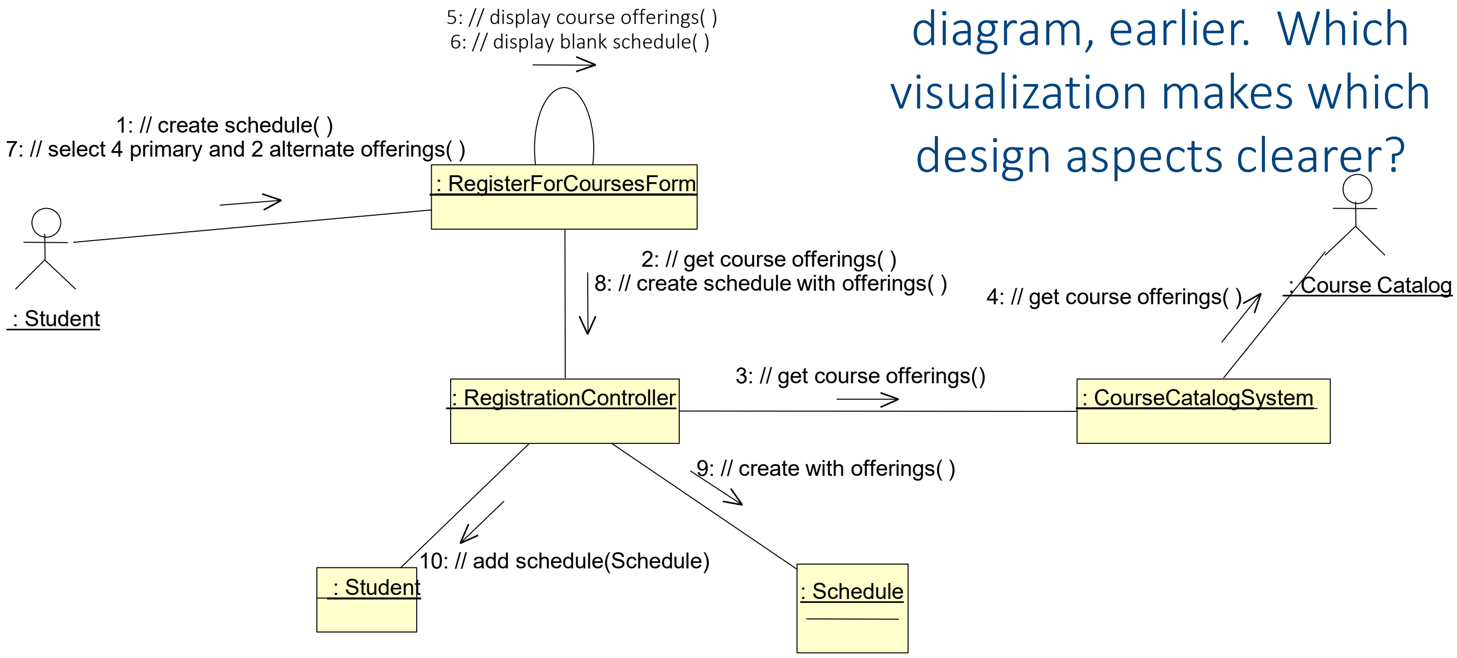
Note: A common mistake is to associate the behavior with the client, instead of the supplier.

Even though the client initiates the behavior and needs it done, it is the supplier that is responsible for carrying out the behavior at the client's request.

Collaboration Diagram for “Register for Courses”

Basic flow, create a schedule portion

Compare this to the equivalent sequence diagram, earlier. Which visualization makes which design aspects clearer?



5: // display course offerings( )
6: // display blank schedule( )
1: // create schedule( )
7: // select 4 primary and 2 alternate offerings( )
: RegisterForCoursesForm
: Student
2: // get course offerings( )
8: // create schedule with offerings( )
4: // get course offerings( )
: Course Catalog
3: // get course offerings()
: RegistrationController
: CourseCatalogSystem
9: // create with offerings( )
10: // add schedule(Schedule)
: Student
: Schedule

Collaboration Diagrams vs. Sequence Diagrams

- Collaboration Diagrams
  - Show relationships (i.e., structure) in addition to interactions
  - Better for visualizing patterns of collaboration
  - Better for visualizing all of the effects on a given object
  - Easier to use for brainstorming sessions
    - CRC (Class, Responsibility, Collaboration) cards

- Sequence Diagrams
  - Show the explicit sequence of messages
  - Better for visualizing overall flow of control
  - Better for real-time specifications and for complex scenarios

Use-Case Analysis - Steps

- Supplement the Use-Case Description

- For each use-case realization
  - Find classes from use-case behavior
  - Distribute use-case behavior to classes

Next

- For each resulting analysis class
  - Describe responsibilities
  - Describe attributes and associations
  - Qualify architectural analysis mechanisms

- Unify analysis classes

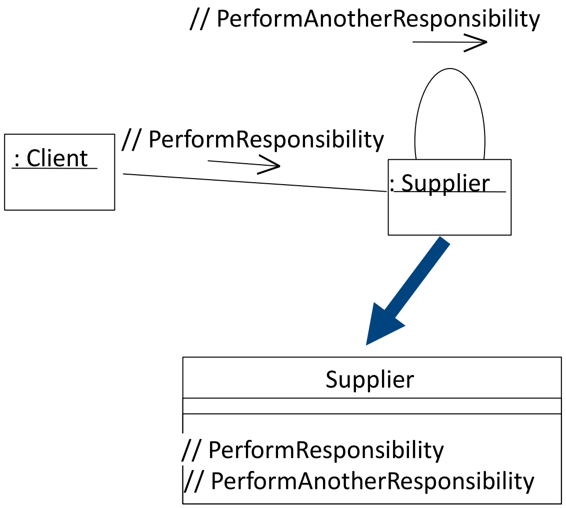
- Checkpoints

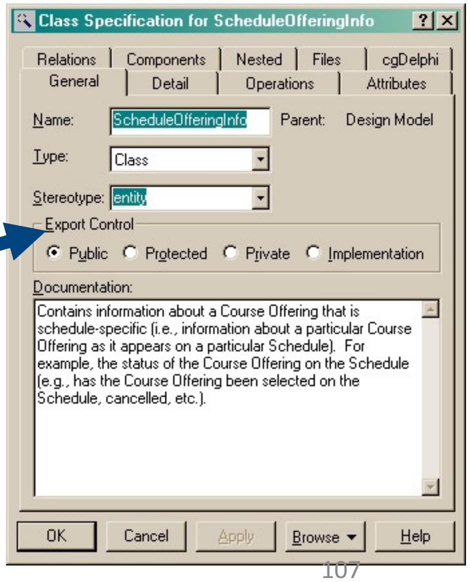
Describe Responsibilities

- A responsibility is a statement of something an object can be asked to provide
  - Actions the object can perform on request
  - Knowledge the object maintains and provides to other objects

- Messages on interaction diagrams define responsibilities
  - Responsibilities evolve into one or more operations on classes in design

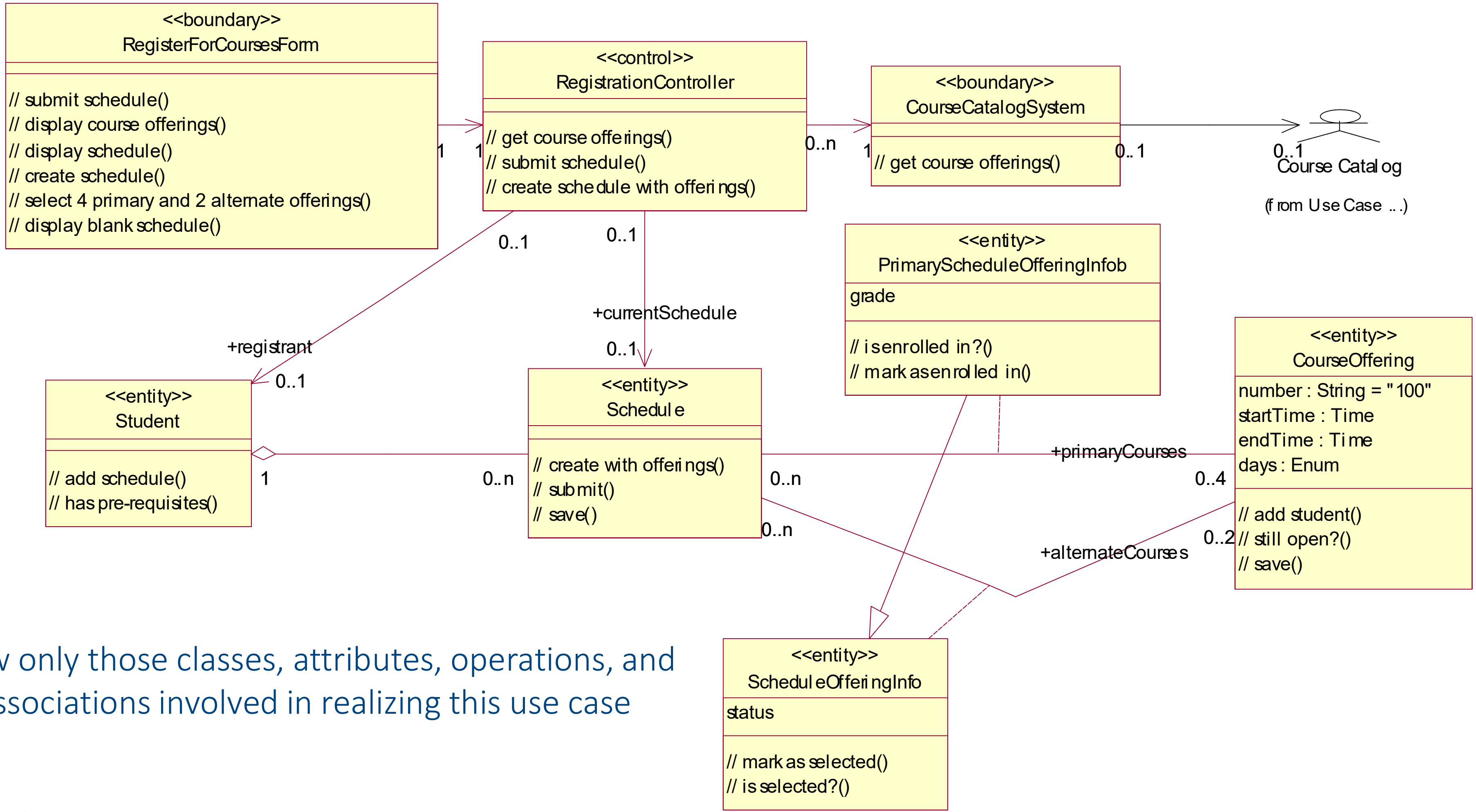
- Document responsibilities in one of two ways
  - As “analysis” operations
    - Class operations with “//” pre-pended as a naming convention
  - As textual description of the analysis class


// PerformAnotherResponsibility
: Client
// PerformResponsibility
: Supplier
Supplier
// PerformResponsibility
// PerformAnotherResponsibility


Class Specification for ScheduleOfferingInfo
Relations
Components
Nested
Files
cgDelphi
General
Detail
Operations
Attributes
OK
Cancel
Apply
Browse
Help

Name: ScheduleOfferingInfo
Parent: Design Model
Type: Class
Stereotype: entity
Export Control
● Public ○ Protected ○ Private ○ Implementation
Documentation:
Contains information about a Course Offering that is schedule-specific (i.e., information about a particular Course Offering as it appears on a particular Schedule). For example, the status of the Course Offering on the Schedule (e.g., has the Course Offering been selected on the Schedule, cancelled, etc.).

“Register for Courses” Realization View of Participating Classes (VOPC)



<<boundary>>
RegisterForCoursesForm

// submit schedule()
// display course offerings()
// display schedule()
// create schedule()
// select 4 primary and 2 alternate offerings()
// display blank schedule()

<<control>>
RegistrationController

// get course offerings()
// submit schedule()
// create schedule with offerings()

<<boundary>>
CourseCatalogSystem

// get course offerings()

Course Catalog
(from Use Case ...)

<<entity>>
PrimaryScheduleOfferingInfob

grade

// is enrolled in?()
// mark as enrolled in()

<<entity>>
Student

// add schedule()
// has pre-requisites()

<<entity>>
Schedule

// create with offerings()
// submit()
// save()

<<entity>>
CourseOffering

number : String = "100"
startTime : Time
endTime : Time
days : Enum

// add student()
// still open?()
// save()

<<entity>>
ScheduleOfferingInfo

status

// mark as selected()
// is selected?()

Show only those classes, attributes, operations, and associations involved in realizing this use case

Maintaining Consistency: What to Look For

- In order of criticality:
    - Redundant responsibilities across classes
    - Disjoint responsibilities within classes
    - Class with one responsibility
    - Class with no responsibilities
    - Better distribution of behavior
    - Class that interacts with many other classes

- A class does everything about X, and only things about X
    - Decompose; separate concerns
    - Integrate; combine all aspects of a single concern

- Update the interaction diagrams to reflect the re-factored classes

Use-Case Analysis - Steps

- Supplement the use-case description

- For each use-case realization
  - Find classes from use-case behavior
  - Distribute use-case behavior to classes

- For each resulting analysis class
  - Describe responsibilities
  - Describe attributes and associations
    - Define attributes
    - Establish associations between analysis classes
    - Describe event dependencies between analysis classes
  - Qualify architectural analysis mechanisms

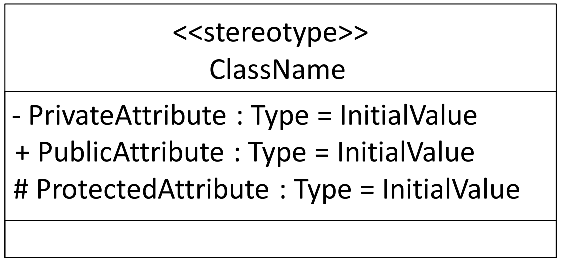
- Unify analysis classes
- Checkpoints

Next

Review: Attributes

- Attributes store information
  - Atomic
  - No responsibilities

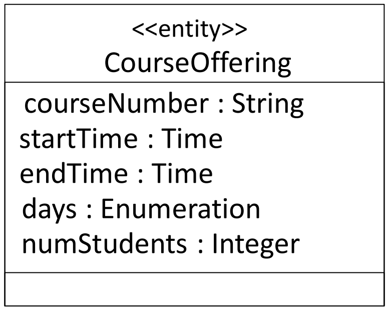
Notation



<<stereotype>>
ClassName

- PrivateAttribute : Type = InitialValue
+ PublicAttribute : Type = InitialValue
# ProtectedAttribute : Type = InitialValue

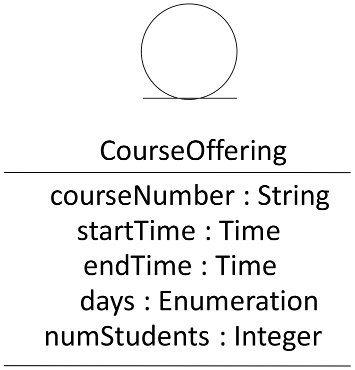
- Attribute name should clearly state what information the attribute holds
  - Supplement with attribute description, where the name is not sufficient



<<entity>>
CourseOffering

courseNumber : String
startTime : Time
endTime : Time
days : Enumeration
numStudents : Integer

- During analysis, attribute types should be from the domain, not from the programming language



CourseOffering

courseNumber : String
startTime : Time
endTime : Time
days : Enumeration
numStudents : Integer

  - In analysis, don’t worry too much about attribute types/signatures
  - But do look for types that should be new classes

Finding Attributes

- Properties/characteristics of identified classes

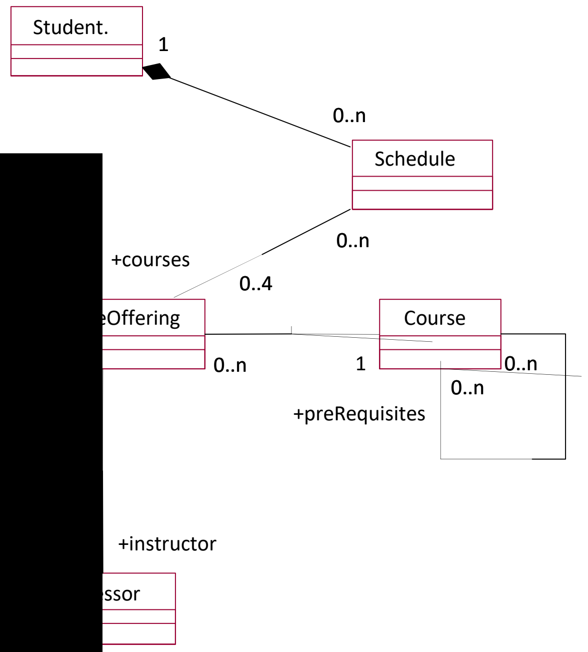
- Information retained by identified classes

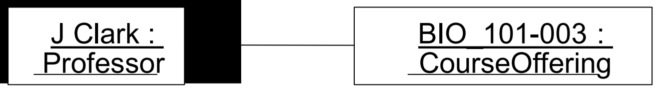
- “Nouns” that did not become classes
    - Information whose value is the important thing
    - Information that is uniquely “owned” by an object
        - No other object refers to it
    - Information that has no behavior beyond get, set, and simple transformations

- Only identify attributes relevant to the domain
    - Avoid interesting but irrelevant characterizations

Review: Associations

[redacted]


Student.
1
0..n
Schedule
0..n
+courses
0..4
eOffering
Course
0..n
1
0..n
0..n
+preRequisites
+instructor
ssor


J Clark : Professor
BIO_101-003 : CourseOffering

(instances) is called a link

Finding Associations

- Every link on an interaction diagram indicate the object classes are associated

- An association of a class to itself is needed when two different objects of the same class need to communicate

- Define association role names

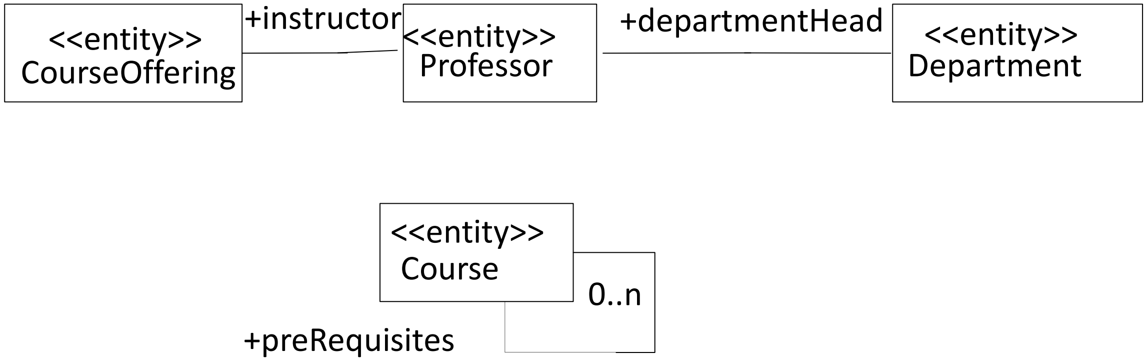
- Define association multiplicities

Finding Associations

- Navigability is optional
    - If defined, make sure messages can flow along navigable paths
- If two objects are bound by a whole-part relationship, the relationship is aggregation
    - When in doubt, use general association
- Only identify associations relevant to the domain
    - Avoid interesting but irrelevant associations

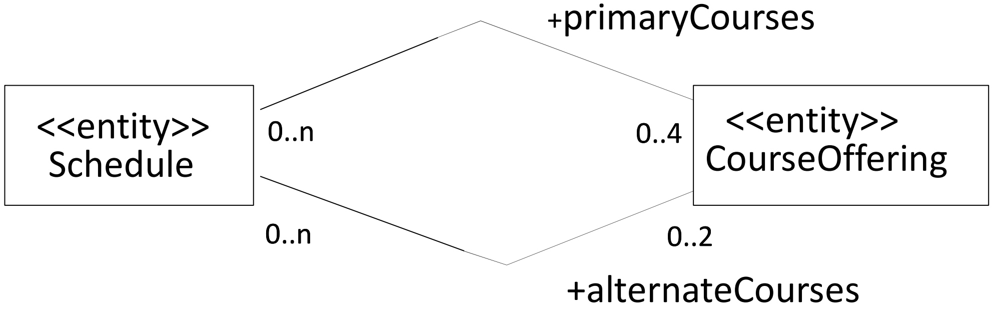
Association Roles

- Association roles define the role an object instance plays for its associated object instance
    - The “face” it presents to its associated object


+instructor
<<entity>>
CourseOffering
<<entity>>
Professor
+departmentHead
<<entity>>
Department
<<entity>>
Course
0..n
+preRequisites

Multiple Associations

- There can be multiple associations between classes
  - Each represents distinct relationships, different roles



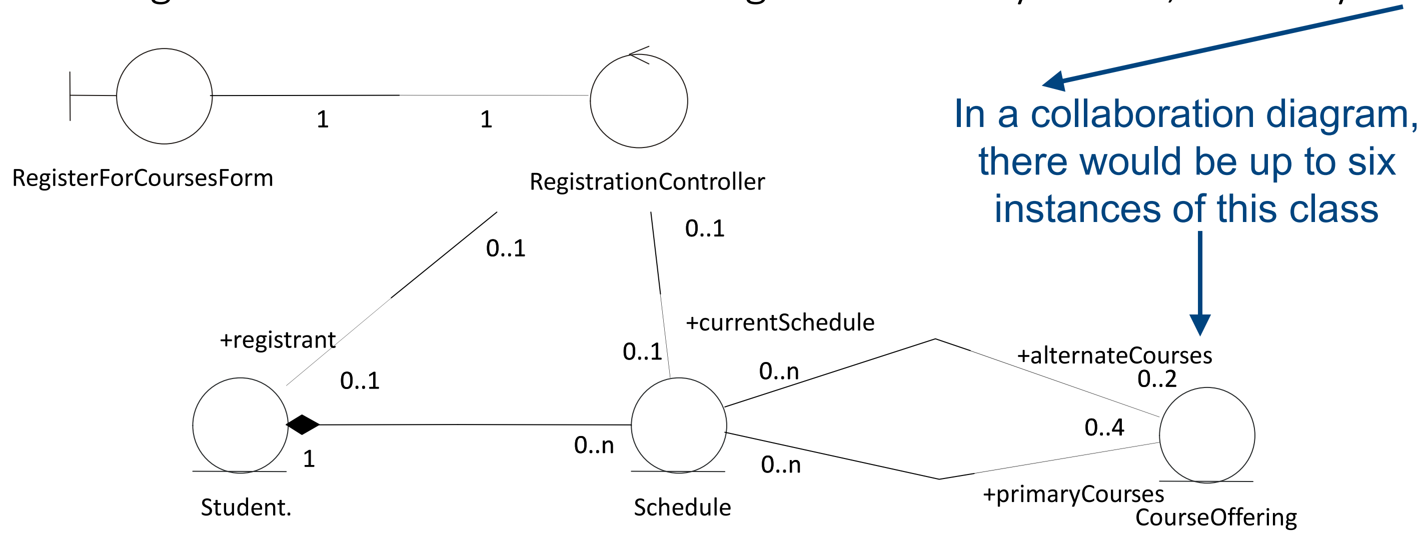
<<entity>>
Schedule

<<entity>>
CourseOffering

+primaryCourses
0..n
0..4
0..n
0..2
+alternateCourses

Finding Relationships

- Draw a class diagram showing the classes participating in a use-case realization interaction diagram
  - View of Participating Classes (VOPC) diagram
  - Each message link in the interaction diagram corresponds to an association in the VOPC diagram
  - Note: the VOPC diagram and the collaboration diagram look very similar, but they are different


1
1
RegisterForCoursesForm
RegistrationController
0..1
0..1
+registrant
0..1
+currentSchedule
0..1
0..n
+alternateCourses
0..2
0..1
1
0..n
0..4
0..n
+primaryCourses
Student.
Schedule
CourseOffering

In a collaboration diagram, there would be up to six instances of this class

Use-Case Analysis - Steps

- Supplement the Use-Case Description
- For each use-case realization
  - Find classes from use-case behavior
  - Distribute use-case behavior to classes
- For each resulting analysis class
  - Describe responsibilities
  - Describe attributes and associations
  - Qualify architectural analysis mechanisms
- Unify analysis classes
- Checkpoints

Next

Describing Analysis Mechanisms

- Collect all the architectural analysis mechanisms in a list

- Map the client classes to the analysis mechanisms

Analysis Class	Analysis Mechanism
Student	Persistence, Security
Schedule	Persistence, Security
CourseOffering	Persistence, Legacy Interface
Course	Persistence, Legacy Interface
RegistrationController	Distribution

- Identify characteristics of the analysis mechanisms for each using class

Example: Describing Analysis Mechanisms (continued)

- Analysis mechanism characteristics

  - Persistence for Schedule class:
    - Granularity: 1 to 10 kbytes per schedule
    - Volume: up to 2,000 schedules
    - Access frequency

      - Create: 500 per day
      - Read: 2,000 per hour
      - Update: 1,000 per day
      - Delete: 50 per day
    - etc.

Use-Case Analysis - Steps

- Supplement the Use-Case Description

- For each use-case realization
  - Find classes from use-case behavior
  - Distribute use-case behavior to classes

- For each resulting analysis class
  - Describe responsibilities
  - Describe attributes and associations
  - Qualify architectural analysis mechanisms

- Unify analysis classes

- Checkpoints

Next

Unify Analysis Classes

- So far, we have identified classes in the context of individual use cases
- Now, unify the results individual results across all the use cases
- Merge classes that define similar behavior or represent the same phenomenon
- Merge entity classes that define the same attributes, even if their behavior is different (merge the behaviors)

Use-Case Analysis - Steps

- Supplement the Use-Case Description

- For each use-case realization
  - Find classes from use-case behavior
  - Distribute use-case behavior to classes

- For each resulting analysis class
  - Describe responsibilities
  - Describe attributes and associations
  - Qualify architectural analysis mechanisms

- Unify analysis classes

Next

- Checkpoints

Review Checkpoints: Analysis Classes

- Are the classes reasonable?

- Does the name of each class clearly reflect the role it plays?

- Does the class represent a single, well-defined abstraction?

- Are all attributes and responsibilities functionally cohesive?

- Does the class offer the required behavior?

- Are all the specific requirements on the class addressed?

- Are there unnecessary attributes or relationships? (remove them!)

Review Checkpoints: Use-Case Realizations

- Have all the main and sub-flows been handled, including exceptional cases?
- Have all the required objects been found?
- Has all behavior been unambiguously distributed to the participating objects?
- Has behavior been distributed to the right objects?
- Where there are several interaction diagrams, are their relationships clear and consistent?

BITS Pilani
Pilani Campus

SOFTWARE DEVELOPMENT PRINCIPLES

Domain model - Conceptual Classes & Associations

Prof A R Rahman

Domain Modeling:
Associations and Attributes

What is a Domain Model?

- Illustrates meaningful conceptual classes in problem domain
- Represents real-world concepts, not software components
- Software-oriented class diagrams will be developed later, during design

How to Make a Domain Model ?

How to create the domain model?

(If we are not readily familiar with the domain.)

1. Find the conceptual classes
   Strategies:
   a. *Reuse or modify an existing model*
   b. *Use a category list*
   c. *Identify noun phrases*

   } explained in the following

2. Draw them as classes in a UML class diagram
3. Add associations and attributes

Use the domain vocabulary; e.g. a model for a library should use names like "Borrower" instead of customer.

- Why:
  Domain modeling helps us to *identify the relevant concepts and ideas of a domain*

- When:
  Domain modeling is done during *object-oriented analysis*

- Guideline:
  Only create a domain model for the tasks at hand

"Curtis'law: [...] Good designs require deep application knowledge."

The Domain Model illustrates noteworthy concepts in a domain.

Domain Model =dt. Analysemodell (Konzept modell)

- The domain model is created during object-oriented analysis to decompose the domain into concepts or objects in the real world

- The model should identify the set of conceptual classes
  (The domain model is iteratively completed.)

- It is the basis for the design of the software

The domain model is also called conceptual model, domain object model or analysis object model.

- Conceptual classes are ideas, things or objects in the domain

A conceptual class has a symbol representing the class, an intension and an extension that defines the set of examples to which the conceptual class applies.

- Domain concepts / conceptual classes are not necessarily software objects as, e.g., in Java, C#, Ruby, Python, ...!

To visualize domain models the UML class diagram notation is used.

- However, no operations are defined in domain models
- Only ...
  - domain objects and conceptual classes
  - associations between them
  - attributes of conceptual classes

The result is a conceptual perspective model.

Visualizing Domain Models

- Exemplified

  (By means of a basic course management system.)

Statements about a Course Management System

- During a semester a lecturer reads one or more lectures
- Sometimes the lecturer is on leave to do research, in this case (s)he does not give a lecture
- A student usually attends one or more lectures, unless (s)he has something better to do
- During the semester there will be several exercises which are meant to be solved by small study groups
- Each student is assigned to one particular study group for the whole semester
- A study group consists of two to three students
- After submission of a solution by a study group it is graded by a tutor/Professor

A class describes a set of objects with the same semantics, properties and behavior.

When used it for domain modeling, it is a visualization of a real world concept

- During a semester a lecturer reads one or more lectures
- A student usually attends one or more lectures
- During the semester there will be several exercises
- Each student is assigned to one particular study group for the whole semester
- It is graded by a tutor

Tutor

Exercise

Semester

Student

Lecture

Lecturer

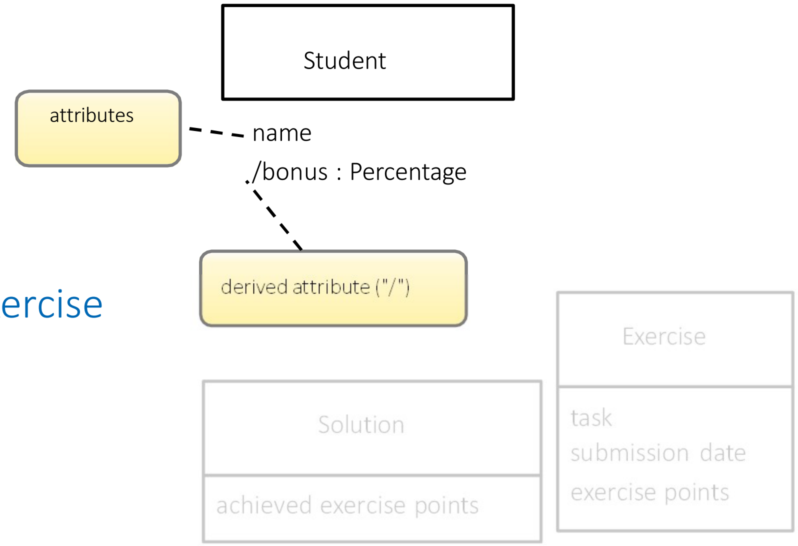
Grade

Study Group

Attributes are logical data values of an object.

It is useful to identify those attributes of conceptual classes that are needed to satisfy the information requirements of the current scenarios under development.

- After submitting a solution it is graded by a tutor
- The bonus is a relative bonus that reflects the relative number of exercise points gained during the semester
- The bonus is derived.

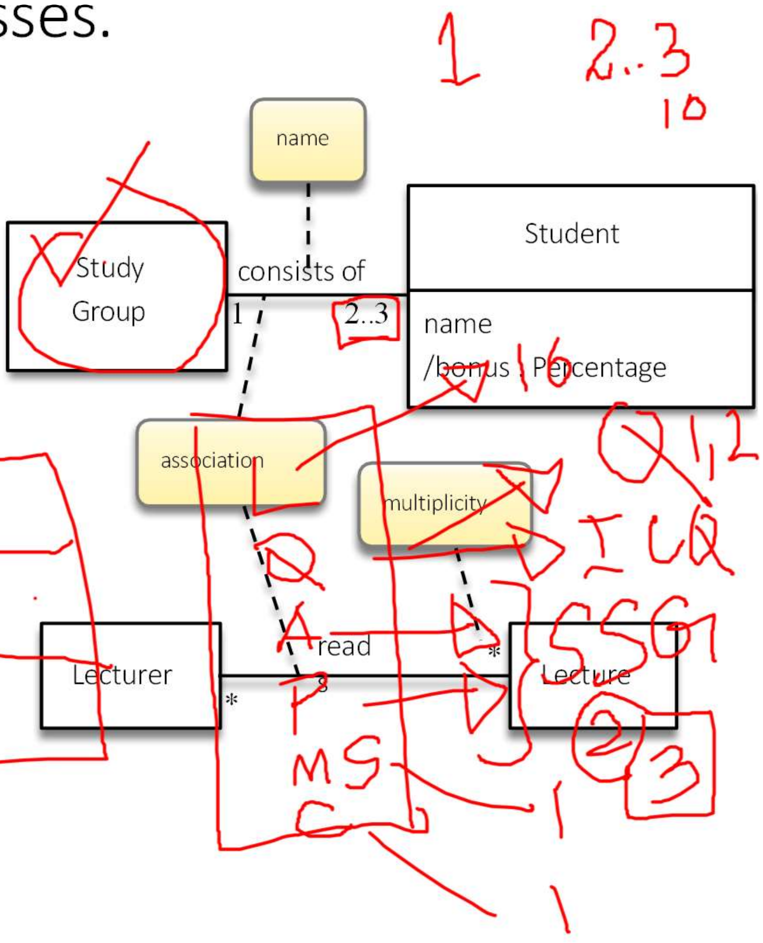

Student
attributes
name
/bonus : Percentage
derived attribute ("/")
Exercise
task
submission date
exercise points
Solution
achieved exercise points

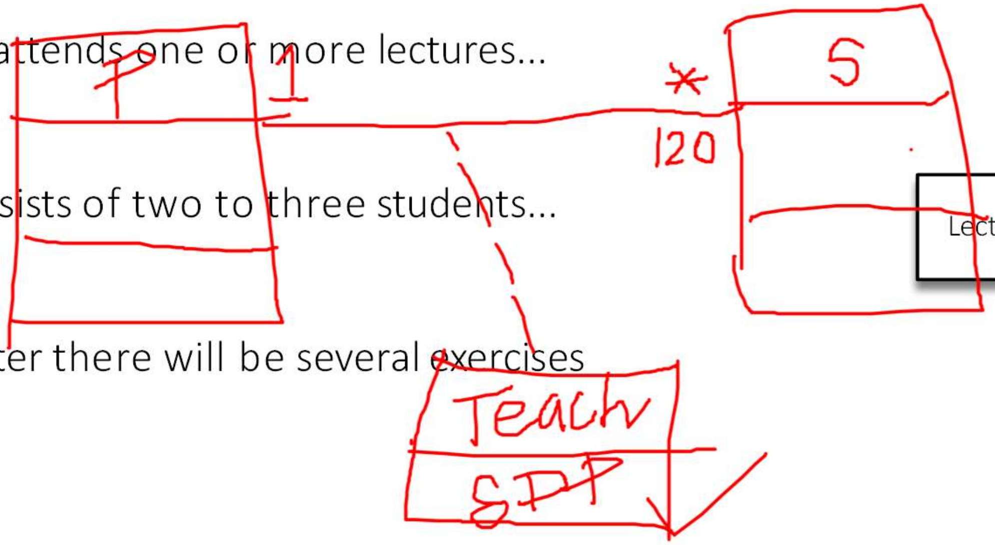
An association is a relationship between classes.

The ends of an association are called roles.

Roles optionally have a multiplicity, name and navigability.

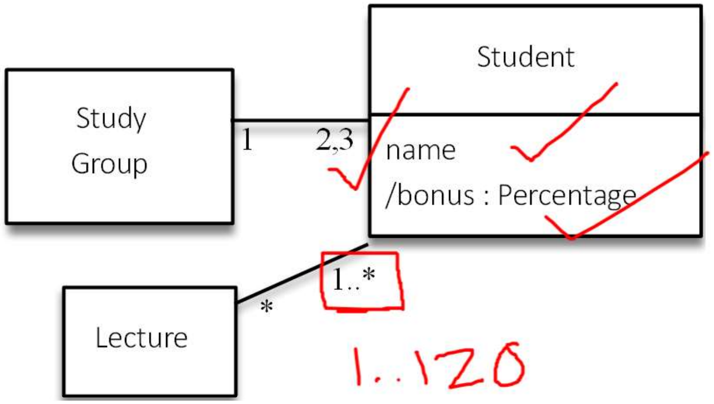
- A lecturer reads one or more lectures...
- A student usually attends one or more lectures...
- A study group consists of two to three students...
- During the semester there will be several exercises
- ...


1
2..3
10
name
Study Group
consists of
1
2..3
Student
name
/bonus : Percentage
16
association
multiplicity
Lecturer
read
*
3
*
Lecture
1,2
ICQ
SSGT
2
3
MS
C
1
1


P
1
*
120
S
Teach
SPP

The multiplicity defines how many instances of a class A can be associated with one instance of a class B at any particular moment.

(e.g., * zero or more; 1..10 between 1 and 10; 1, 2 one or two)

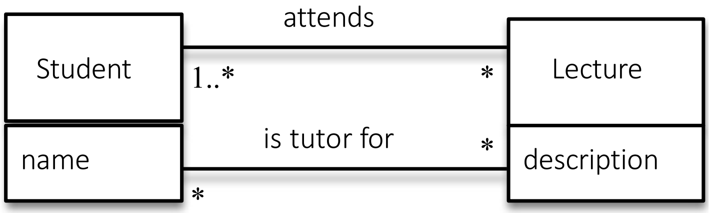

Student
name
/bonus : Percentage
Study Group
1
2,3
Lecture
*
1..*
1..120

- A student usually attends one or more lectures, unless (s)he has something better to do
- A study group consists of two to three students
- ...

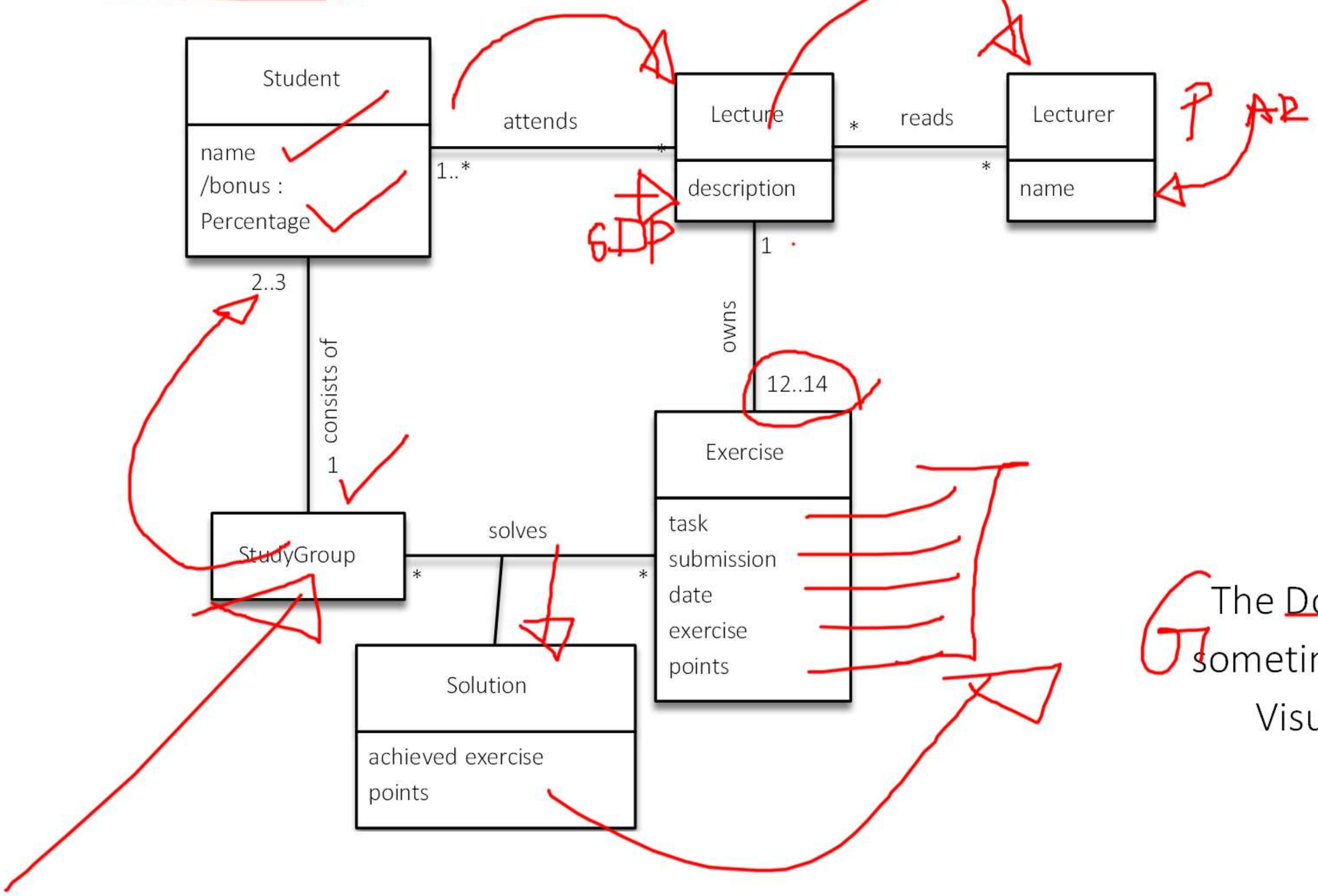
Two classes can have multiple associations.

- A student usually attends one or more lectures, unless the student has something better to do

- A study group consists of two to three students; after submitting a solution it is graded by a tutor who is also a student

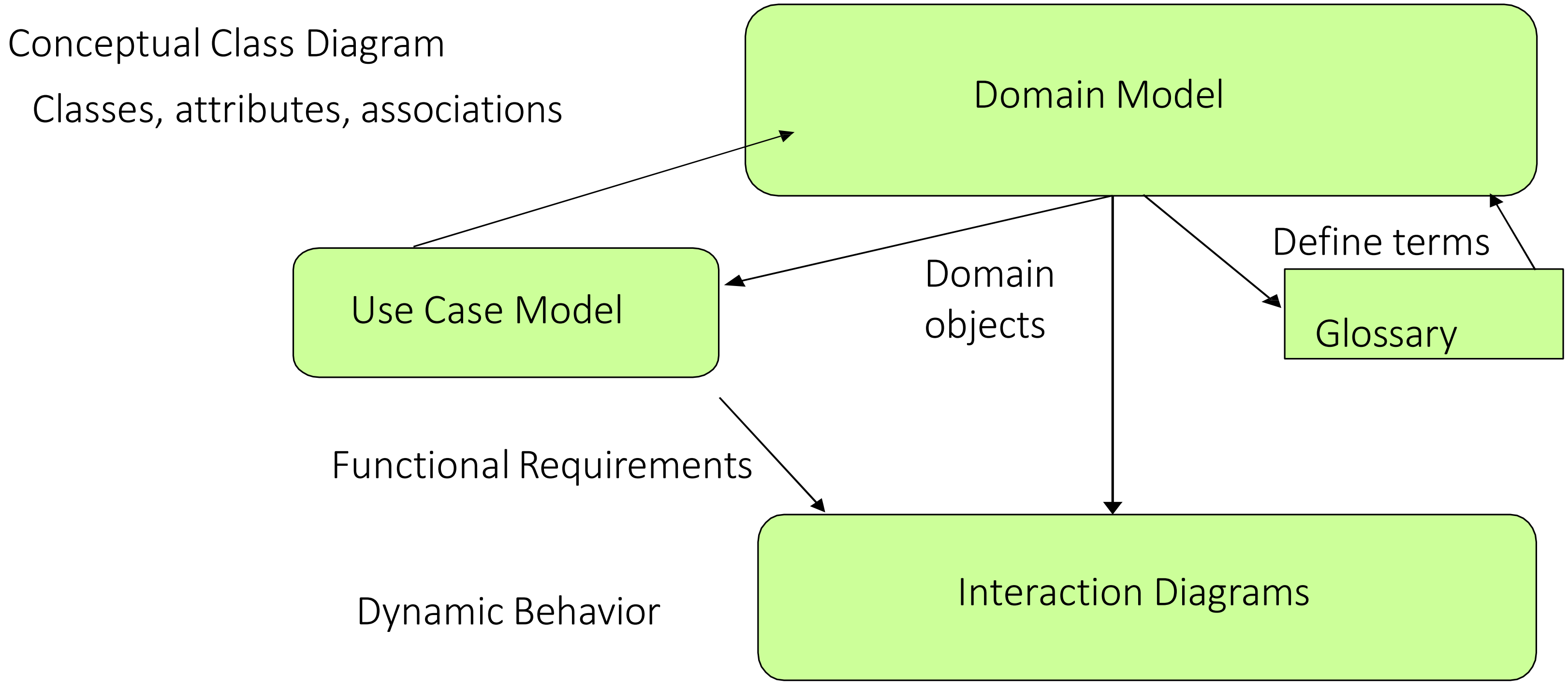

attends
Student
1..*
*
Lecture
is tutor for
*
name
description
*

A preliminary domain model for a course management system.


Student
name
/bonus :
Percentage
attends
1..*
*
Lecture
description
*
reads
*
Lecturer
name
P AR
GDP
1
owns
12..14
2..3
consists of
1
StudyGroup
solves
*
*
Exercise
task
submission
date
exercise
points
Solution
achieved exercise
points

The Domain Model is sometimes also called a Visual Dictionary.

Domain Model Relationships


Conceptual Class Diagram
Classes, attributes, associations
Domain Model
Use Case Model
Domain objects
Define terms
Glossary
Functional Requirements
Dynamic Behavior
Interaction Diagrams

What do you learn about when and how to create these models?

Why do a domain model?

- Gives a conceptual framework of the things in the problem space
- Helps you think – focus on semantics
- Provides a glossary of terms – noun based
- It is a static view - meaning it allows us convey time invariant business rules
- Foundation for use case/workflow modelling
- Based on the defined structure, we can describe the state of the problem domain at any time.

Features of a domain model

- Domain classes – each domain class denotes a type of object.
- Attributes – an attribute is the description of a named slot of a specified type in a domain class; each instance of the class separately holds a value.
- Associations – an association is a relationship between two (or more) domain classes that describes links between their object instances.

  Associations can have roles, describing the multiplicity and participation of a class in the relationship.
- Additional rules – complex rules that cannot be shown with symbolically can be shown with attached notes.

Domain classes?

- Each domain class denotes a type of object.
- It is a descriptor for a set of things that share common features.
- Classes can be:-
- *Business objects* - represent things that are manipulated in the business e.g. *Order*.
- *Real world objects* – things that the business keeps track of e.g. *Contact*, *Site*.
- *Events that transpire* - e.g. *sale* and *payment*.

1MP 1
1SW 2
1PW 3

How to Identify Domain Classes

- Reuse an existing domain model
  - There are many published, well-crafted domain models.

- Use a conceptual class category list
  - Make a list of all candidate conceptual classes

- Identify noun phrases
  - Identify nouns and phrases in textual descriptions of a domain (use cases, or other documents)

Conceptual Class Category List

- Physical or tangible objects
  - Register, Airplane
- Specifications, or descriptions of things
  - ProductSpecification, FlightDescription
- Places
  - Store, Airport
- Transactions
  - Sale, Payment, Reservation
    - Transaction items
      - SalesLineItem
- Roles of people
  - Cashier, Pilot
- Containers of other things
  - Store, Hangar, Airplane
    - Things in a container
      - Item, Passenger
- Computer or electro mechanical systems
  - CreditPaymentAuthorizationSystem, AirTrafficControl
- Catalogs
  - ProductCatalog, PartsCatalog
- Organizations
  - SalesDepartment, Airline

Where identify conceptual classes from noun phrases (NP)

- Vision and Scope, Glossary and Use Cases are good for this type of linguistic analysis
- However:
- Words may be ambiguous or synonymous
- Noun phrases may also be attributes or parameters rather than classes:
  - If it stores state information or it has multiple behaviors, then it’s a class
  - If it’s just a number or a string, then it’s probably an attribute

e.g. From NPs to classes or attributes

Consider the following problem description, analyzed for Subjects, Verbs, Objects:

The ATM verifies whether the customer's card number and PIN are correct.
S**C** V **R** O O **A** O**A**
If it is, then the customer can check the account balance, deposit cash, and withdraw cash.
S **R** V O **A** V O **A** V O**A**
Checking the balance simply displays the account balance.
S**M** O **A** V O**A**

Depositing asks the customer to enter the amount, then updates the account balance.

S **M** V O**R** V O**A** V O**A**

Withdraw cash asks the customer for the amount to withdraw; if the account has enough cash,
S **M** O**A** V O **R** O**A** V S**C** V O**A**
the account balance is updated. The ATM prints the customer's account balance on a receipt.
O **A** V S**C** V O**A** O

Analyze each **subject** and **object** as follows:

- Does it represent a person performing an action? Then it’s an actor, ‘R’.

- Is it also a verb (such as ‘deposit’)? Then it may be a method, ‘M’.

- Is it a simple value, such as ‘color’ (string) or ‘money’ (number)?

  Then it is probably an attribute, ‘A’.

- Which NPs are unmarked? Make it ‘C’ for class.

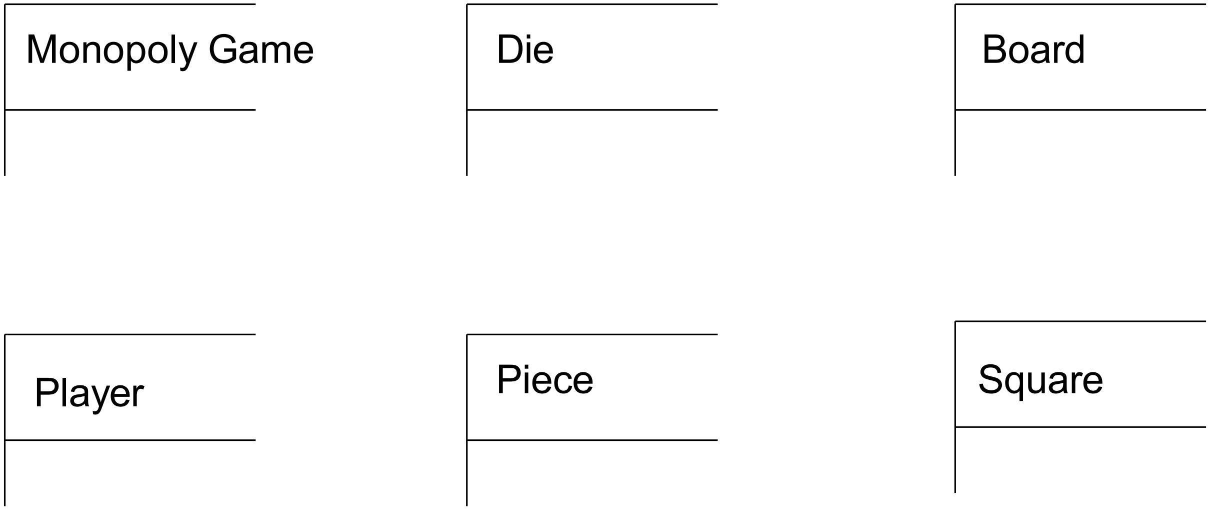
- Verbs can also be classes, for example:

  Deposit is a class if it retains state information

Steps to create a Domain Model

1. Identify candidate conceptual classes
2. Draw them in a UML domain model
3. Add associations necessary to record the relationships that must be retained
4. Add attributes necessary for information to be preserved
5. Use existing names for things, the vocabulary of the domain

Monopoly Game domain model (first identify concepts as classes)



Monopoly Game

Die

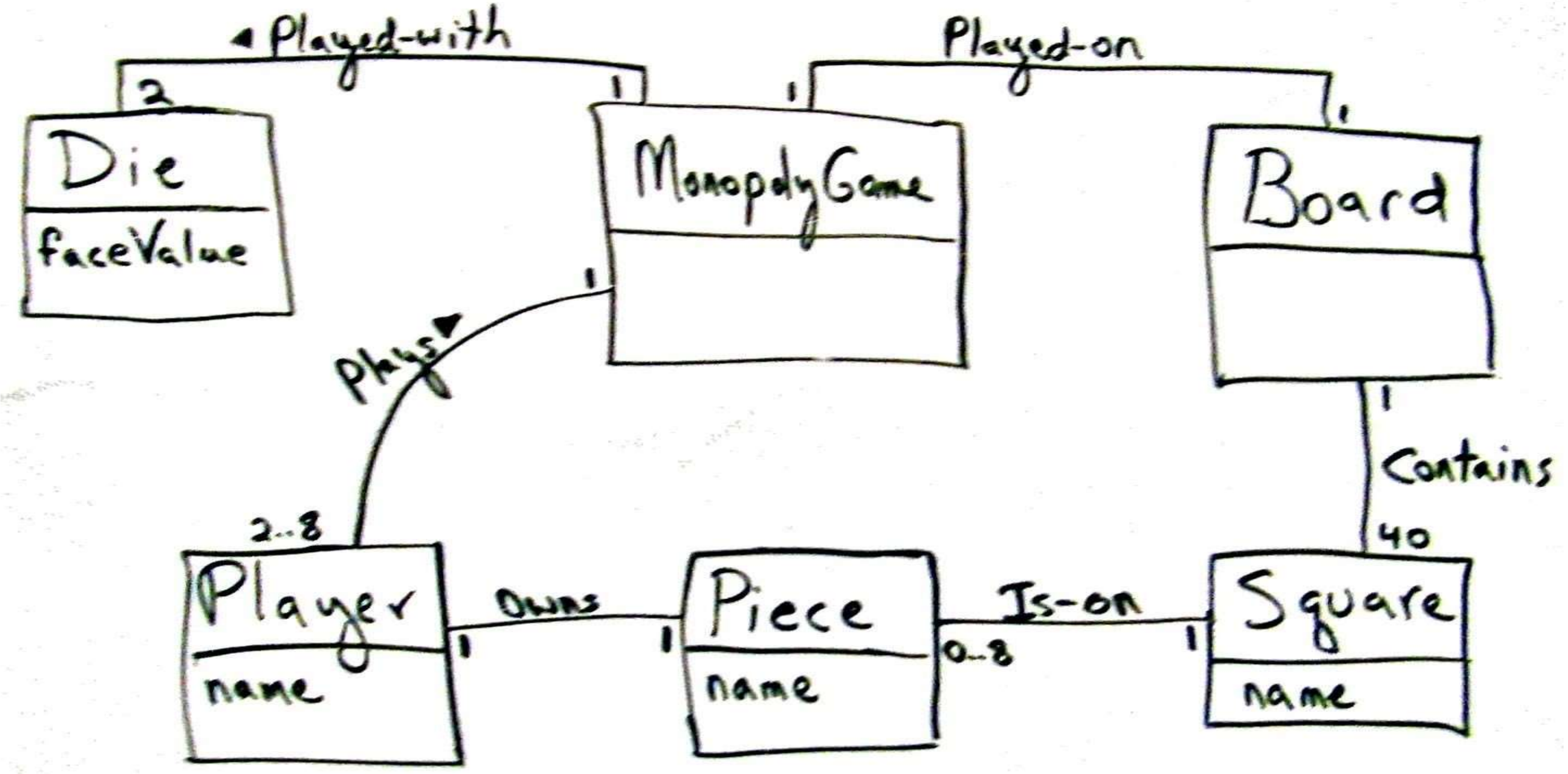
Board

Player

Piece

Square

Monopoly Game domain model Larman, Figure 9.28


◂ Played-with
2
1
Die
faceValue
MonopolyGame
1
Played-on
1
1
Board
1
Contains
40
Plays ▸
1
2..8
Player
name
Owns
1
1
Piece
name
Is-on
0..8
1
Square
name

Class names

- **Class Name** creates the vocabulary of our analysis
  - Use nouns as class names, think of them as simple agents
  - Verbs can also be made into nouns, if they are maintain state
  - E.g., “reads card” suggests **CardReader**, managing bank cards
- Use pronounceable names:
  - If you cannot read aloud, it is not a good name
- Use capitalization to initialize Class names and demarcate multi-word names
  - E.g., CardReader ✓ rather than CARDREADER or card_reader ✗
  - Why do most OO developers prefer this convention?

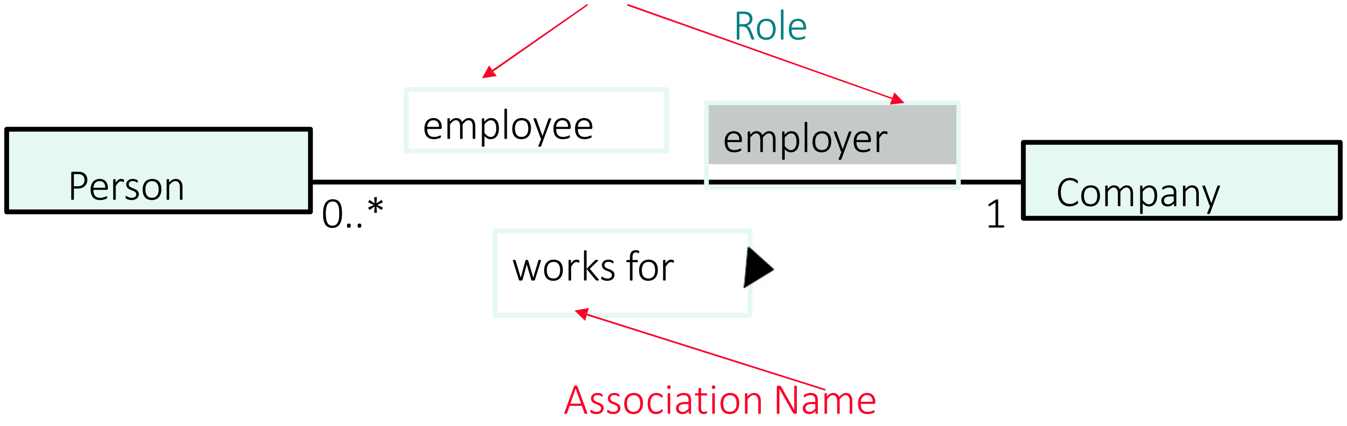
Class names

- Avoid obscure, ambiguous abbreviations
    - E.g., is TermProcess something that terminates
    - or something that runs on a terminal?

- Try *not* to use digits within a name, such as CardReader2
    - Better for instances than classes of objects

Associations

- A link between two classes (“has a”)
  - Typically modeled as a member reference
  - Notation from Extended Entity Relation (EER) models
- A Person works for a Company



Role
employee
employer
Person
0..*
1
Company
works for
Association Name

- Role names and multiplicity at association ends
- Direction arrow to aid reading of association name

Adding Association

- An association is a relationship between classes that indicates some meaningful and interesting connection.

- In the UML, associations are defined as “the semantic relationship between two or more classifiers that involve connections among their instances.”

Structure (association) analysis

- Lines connecting classes
- In UML, simple line is an **association**
  - Decorations for multiplicity, role names, constraints
- Aggregations and composition:
  - Arrow denotes navigability
  - A **black-filled** diamond denotes a **composition**
    - a part, **unique** to this whole
  - A **white-empty** diamond denotes an **aggregation**
    - a part, but not unique to this whole

Common Associations

- A is subpart/member of B. (SaleLineItem-Sale)
- A uses or manages B. (Cashier –Register, Pilot- airplane)
- A communicates with B. (Student -Teacher)
- A is transaction related to B. (Payment -Sale)
- A is next to B. (SaleLineItem-SaleLineItem)
- A is owned by B. (Plane-Airline)
- A is an event related to B. (Sale-Store)

Roles and Multiplicity

- Each end of an association is called a role.

- Multiplicity defines how many instances of a class A can be associated with one instance of a class B.

- e.g.: a single instance of a Store can be associated with “many”(zero or more) Item instances.

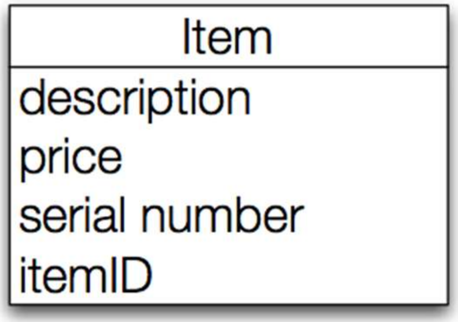
Adding Attributes

- An attribute is a logical data value of an object.
- Include the following attributes in a domain model:
- Those for which the requirements suggest a need to remember information.
- An attribute can be a more complex type whose structure is unimportant to the problem, so we treat it like a simple type
- UML Attributes Notation: Attributes are shown in the second compartment of the class box

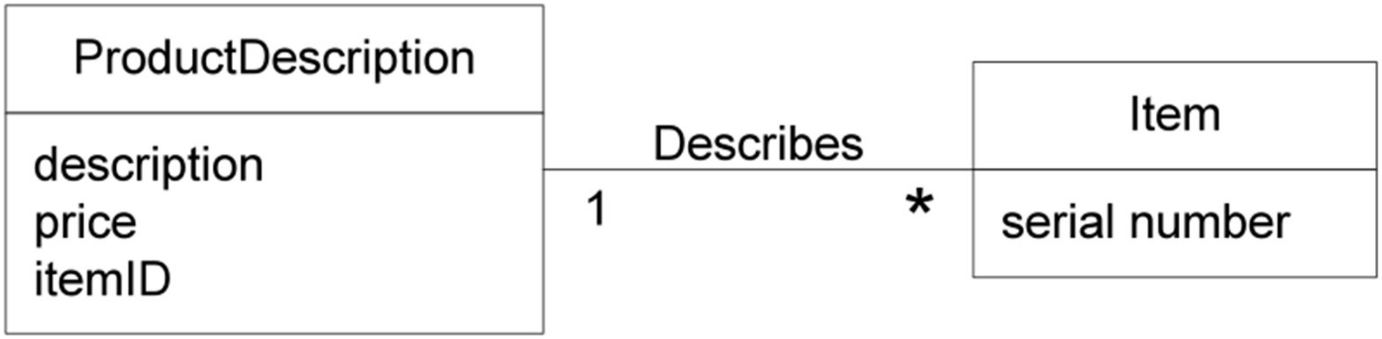
Description Classes

- A description class contains information that describes something else (e.g., *ProductDescription*)

- Example: An *Item* instance represents a physical item in a store
  - Item data only recorded within *Item* instances
  - When a real-world item is sold, we remove the software *Item* goes away…
  - Problem: last Item sold, how much for the Item?
  - Duplication also a problem…



Solution: Use A Description Class


ProductDescription
description
price
itemID
Describes
1
*
Item
serial number

1. When information must be retained independent of existence of instances of the described item
2. When deleting the described item could result in information loss
3. When it reduces redundant information

Creating a Domain Model

- Identify Candidate Conceptual classes
- Draw them in the initial Domain Model
- Add requisite associations to record key relationships
- Add attributes to preserve relevant information

Creating a Domain Model

- Apply existing Analysis Patterns
  - Use existing names for things, the vocabulary of the domain
  - Exclude irrelevant features
- Rules of thumb for Conceptual Classes
  - If it takes up space, then it is probably a conceptual class.
  - If you can’t think of a thing as a number or text, then it is probably a conceptual class.

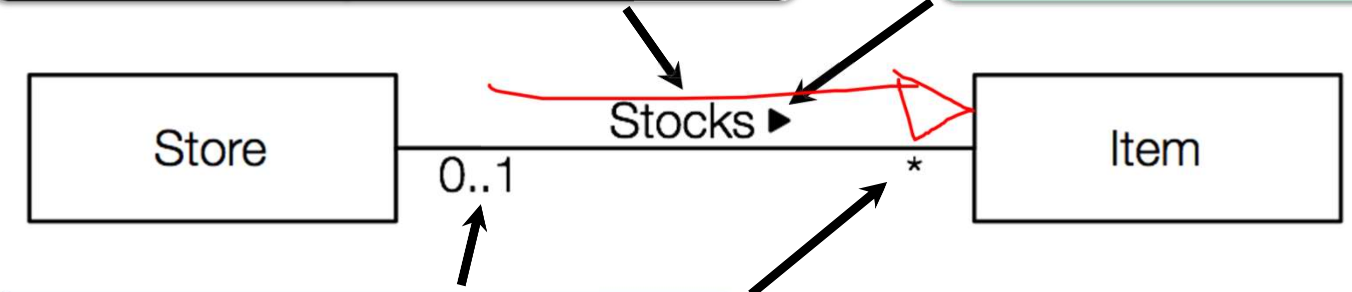
Associations

- A relationship between classes that indicate some meaningful connection between instances of the classes
  - Says that we need some memory of the relationship
  - A memory in the real world, not a software need
- Not about data flows, foreign key relationships, instances variables, or software pointers

Association Notation

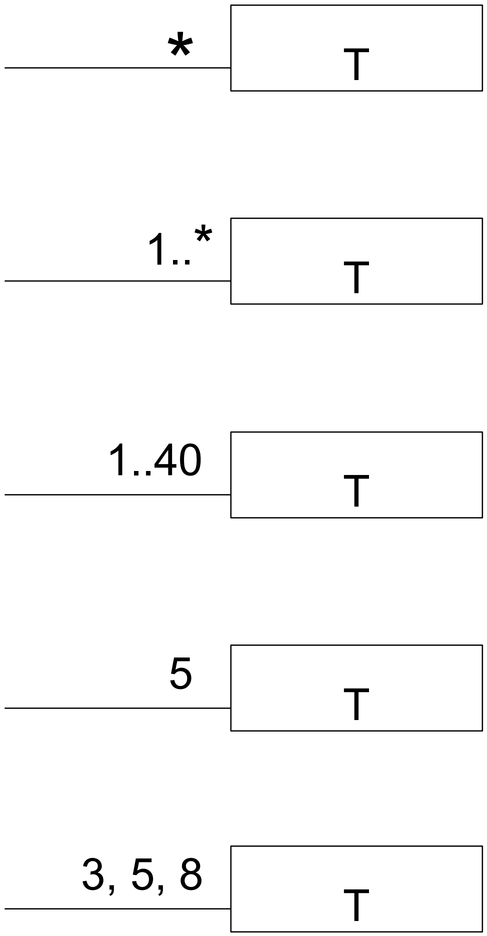
Association name:
- ✓ Use verb phrase
- ✓ Capitalize
- ✓ Typically camel-case or hyphenated
- ✓ Avoid “has”, “use

Reading direction: Can exclude if association reads left-to-right or top-to-bottom



Multiplicity (Cardinality):
- ✓ ‘*’ means “many”
- ✓ x..y means from x to y inclusively

Multiplicity (AKA Cardinality)



zero or more;
"many"

one or more

one to 40

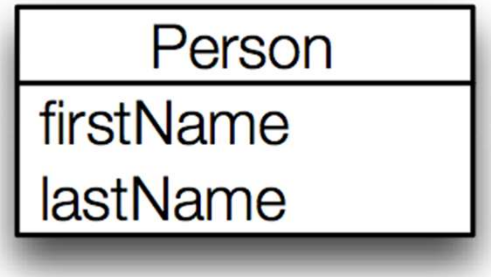
exactly 5

exactly 3, 5, or 8

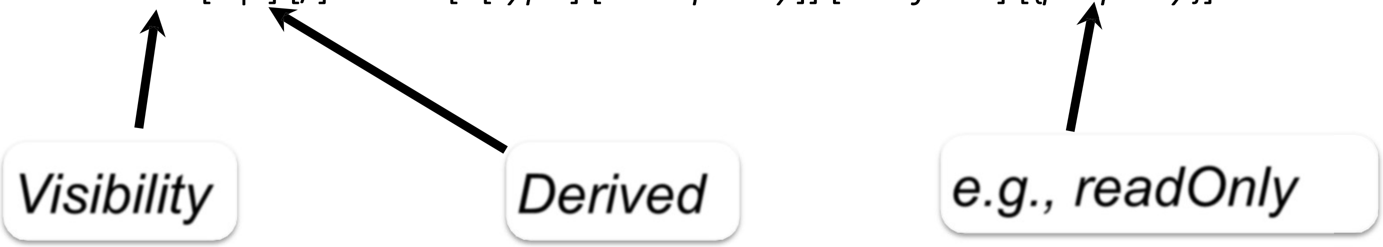
Common Association Lists

Association Category	POS Examples
A is a transaction related to another transaction *B*	CashPayment PaysFor Sale
A is a line item of a transaction *B*	SalesLineItem ContainedIn Sale
A is known/logged/recorded in/on B	Sale CapturedOn Register
…	…

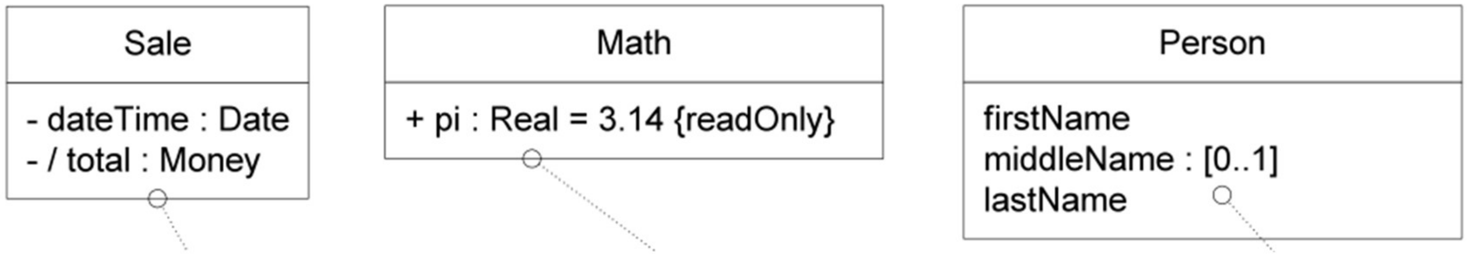
Attributes


Person
firstName
lastName

- Include attributes that the requirements suggest need to be remembered
- Notation ("[ ]"indicate optional parts):
  - [+|-] [/] *name* [: [*type*] [*multiplicity*]] [= *default*] [{*property*}]


Visibility
Derived
e.g., readOnly

Attribute Examples

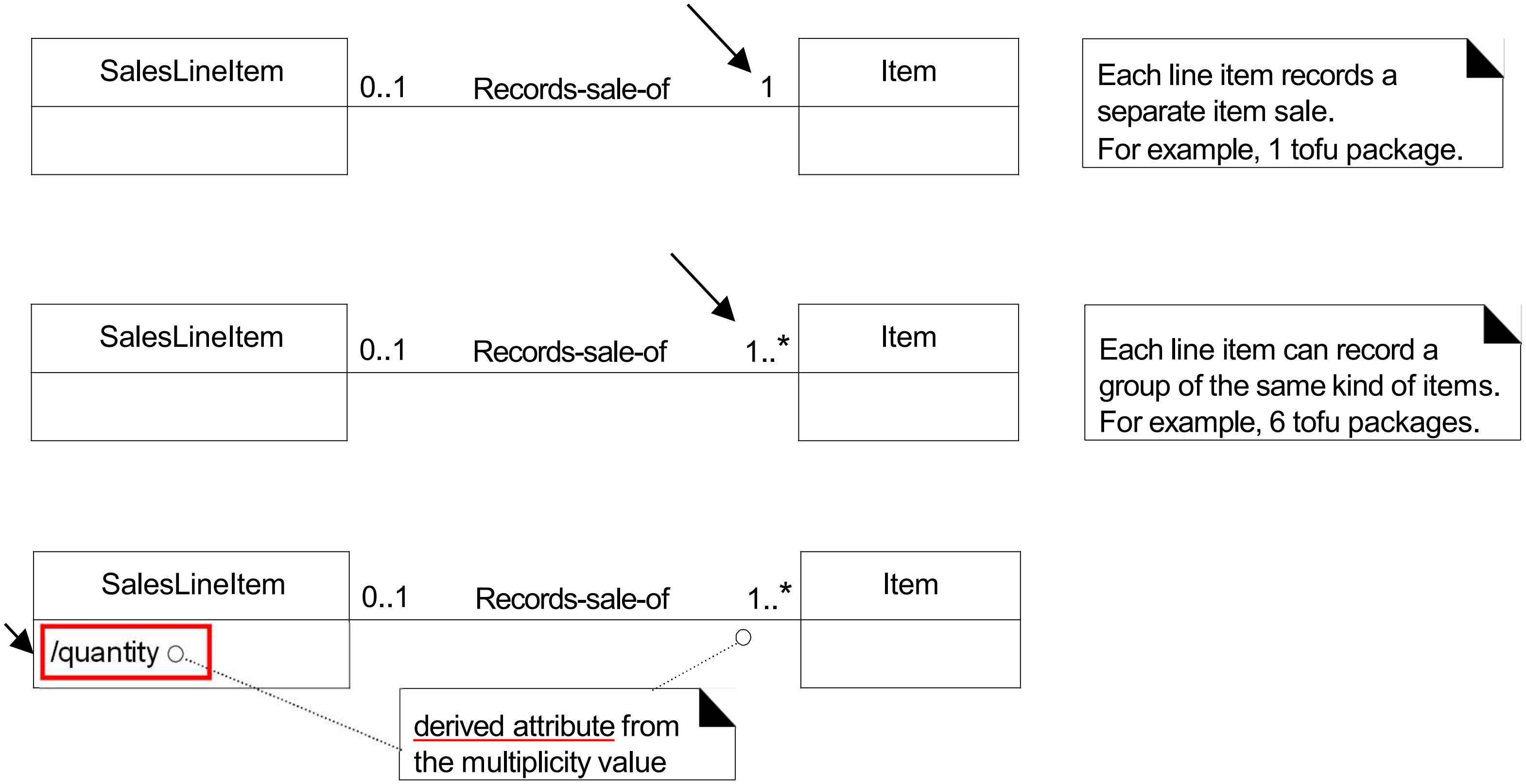


Sale
- dateTime : Date
- / total : Money
Math
+ pi : Real = 3.14 {readOnly}
Person
firstName
middleName : [0..1]
lastName

- What does each part mean?

Note: While visibility is possible in Domain Models, use it sparingly.

Recording Quantity of Items


SalesLineItem
0..1
Records-sale-of
1
Item
SalesLineItem
0..1
Records-sale-of
1..*
Item
SalesLineItem
/quantity
0..1
Records-sale-of
1..*
Item

Each line item records a separate item sale.
For example, 1 tofu package.

Each line item can record a group of the same kind of items.
For example, 6 tofu packages.

derived attribute from the multiplicity value

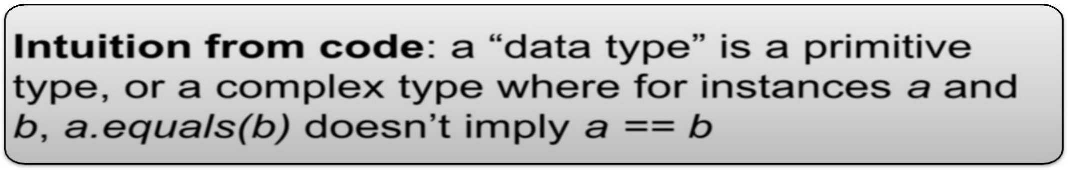
In Domain Model, Use Data Type Attributes

- Primitive data types:
  - Boolean, String, Real, Integer, …

- Sometimes more complex, but not domain specific:
  - Address, Color, Phone Number, …

- If it’s domain specific, use a class and association

Intuition from code: a “data type” is a primitive type, or a complex type where for instances *a* and *b*, *a.equals(b)* doesn‟t imply *a == b*

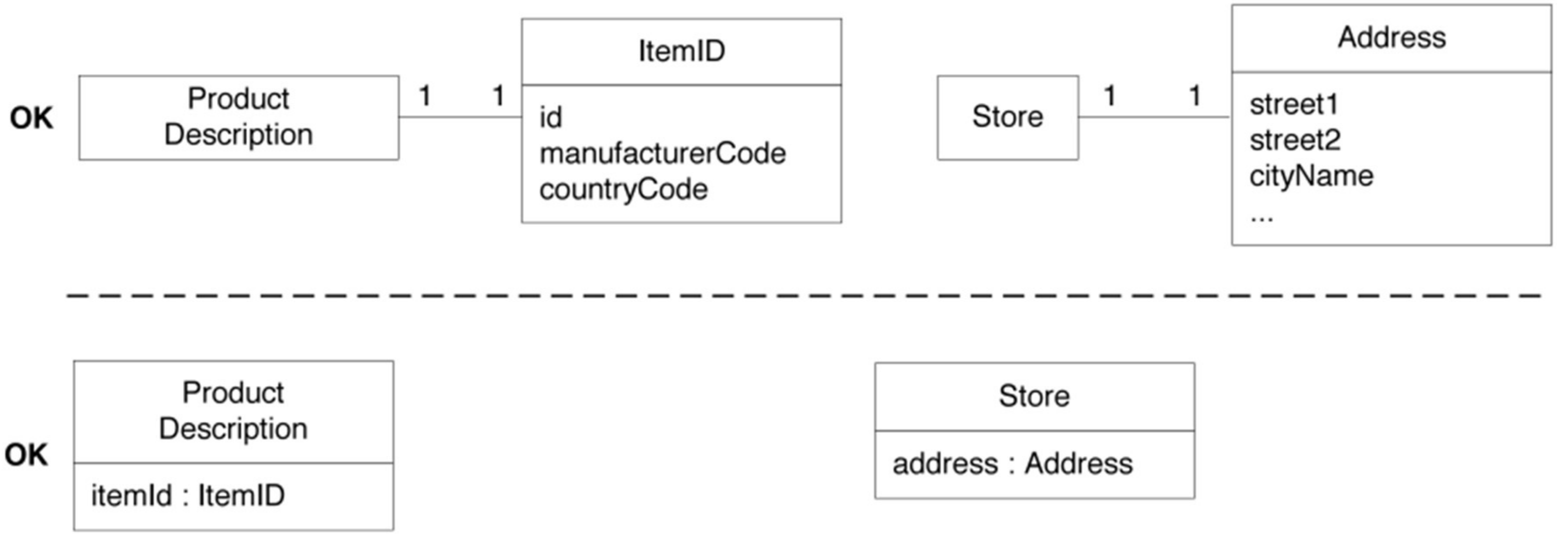


Intuition from code: a “data type” is a primitive type, or a complex type where for instances *a* and *b*, *a.equals(b)* doesn’t imply *a == b*

Create Your Own Complex Data Type When

- It has attributes of its own
- There are operations associated with it (e.g., validation)
- It's a quantity with a unit

Showing Data Type Attributes


OK
Product Description
1
1
ItemID
id
manufacturerCode
countryCode
Store
1
1
Address
street1
street2
cityName
...
OK
Product Description
itemId : ItemID
Store
address : Address

Choose the representation that best communicates with the stakeholders

Domain Model Guidelines, Summarized

- Classes first, then associations and attributes

- Use existing models, category lists, noun phrases

- Include “report objects”, like Receipt, if they’re part of the business rules

- Use terms from the domain

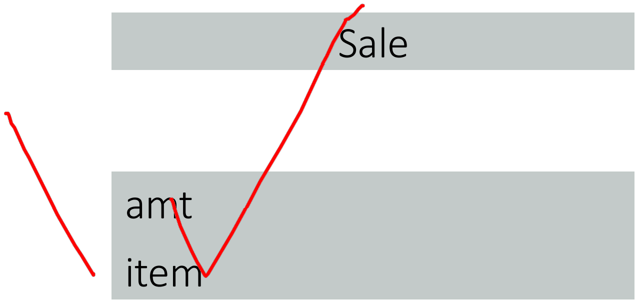
- Don’t send an attribute to do a conceptual class’s job
- Use description classes to remember information independent of instances and to reduce redundancy
- Use association for relationship that must be remembered

Domain Model Guidelines, Summarized

- Be parsimonious with associations
- Name associations with verb phrases, not “has” or “uses”
- Use common association lists
- Use attributes for information that must be remembered
- Use data type attributes
- Define new data types for complex data
- Communicate with stakeholders

A Domain Model is Conceptual, not a Software Artifact

Conceptual Class:

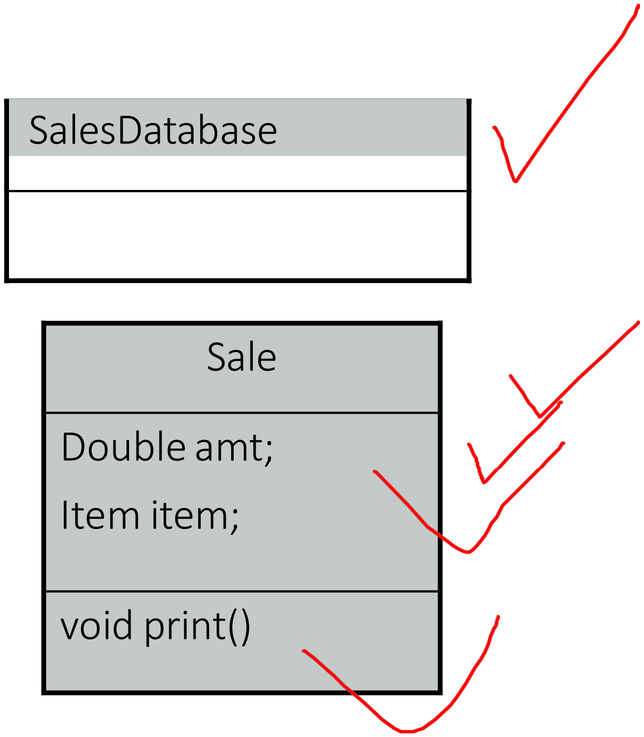


Sale

amt
item

vs.

Software Artifacts:



SalesDatabase

Sale

Double amt;
Item item;

void print()

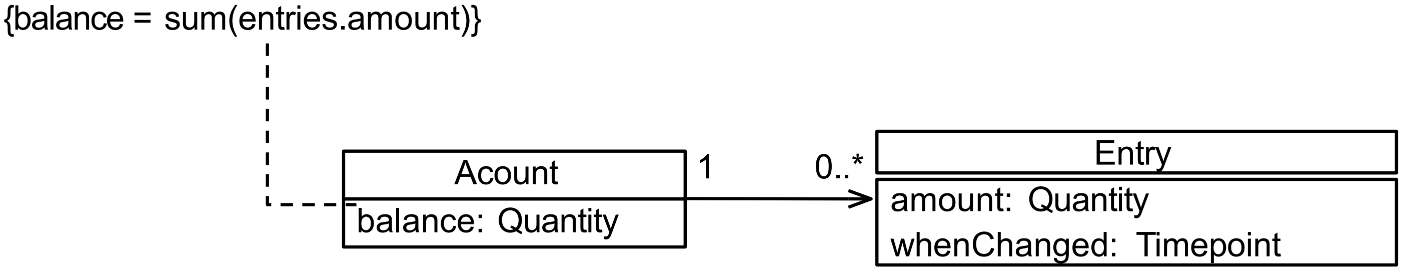
What's the difference?

Excerpt Of the Domain Model For the POS System

?

Which are noteworthy domain concepts / domain objects?

Example: existing model for bank accounts and associated entries.


{balance = sum(entries.amount)}
Acount
balance: Quantity
1
0..*
Entry
amount: Quantity
whenChanged: Timepoint

[One possible source]

Martin Fowler; Analysis Patterns - Reusable Object Models; Addison-Wesley, 1997

How to create the domain model?
Use a category list to find the conceptual classes.

Conceptual Class Category	Classes (for the POS system)
Business transactions...	Sale, Payment
Transaction line items...	SalesLineItem
Product or service related to a transaction or transaction line item.	Item
Where is the transaction recorded?	Register
Roles of people or organizations related to the transaction; actors in use cases.	Cashier, Customer, Store
Place of transactions.	Store
Noteworthy events, often with a time or place that needs to be remembered.	Sale, Payment
...	...

How to create the domain model?
Identify noun phrases to find the conceptual classes(linguistic analysis).

- Identify the nouns and noun phrases in textual descriptions of a domain and consider them as candidate conceptual classes or attributes

- A mechanical noun-to-class mapping isn't possible; words in natural languages are ambiguous;

- i.e. the same noun can mean multiple things and multiple nouns can actually mean the same thing

"Use Cases" are also an excellent source for identifying conceptual classes.

How to create the domain model?
Identify noun phrases to find the conceptual classes.

- Process Sale: A customer arrives at a checkout with items to purchase.
- The cashier uses the POS system to record each item.
- The system presents a running total and line-item details.
- The customer enters payment information, which the system validates and records.
- The system updates the inventory.
- The customer receives a receipt from the system and then leaves the store with the items.

Identified candidate conceptual classes:
Customer, Item, Cashier, Store, Payment, Sales Line Item, Inventory, Receipt, Sale.

Should Receipt be in the domain model?

Guidelines when to include a candidate conceptual class that reports Information into the domain model

- In general, it is not useful since all information is derived or duplicated from other sources
- If it has a specific semantics w.r.t. the business, then it should be included

Identified candidate conceptual classes:
Customer, Item, Cashier, Store, Payment, Sales Line Item, Inventory, Receipt, Sale.

A Receipt is just a report of a sale and a payment

Guidelines when to include a candidate conceptual class that reports Information into the domain model

- “Facts”:
  - Identified candidate conceptual classes: … Receipt, ….
  - A receipt is just a report of a sale and a payment...
- Should Receipt be in the domain model?

Guidelines when to include a candidate conceptual class that reports Information into the domain model

- Well, it depends….
  - … if we just consider the current scenario then receipt should not be part of the domain model; a receipt is just a report
  - … if we also consider how to handle returns then a receipt represents an important concept on its own

Domain Requirement

- How about legal restrictions...?

Guidelines when to include a description class into the domain model

- A description class contains information that describes something else

  Examples:

  - a product description records the price, picture and text of an item

  - a flight description contains information about the flight (number) and the source and target destinations

Guidelines when to include a description class into the domain model

A description class should be added to the domain model when:

- There needs to be a description about an item or service, independent of the current existence of any examples of those items or services
- Deleting instances of things they describe results in a loss of information that needs to be maintained, but was incorrectly associated with the deleted thing
- It reduces redundant or duplicated information

Point of Sale System (POS) [1]

Basic Flow:

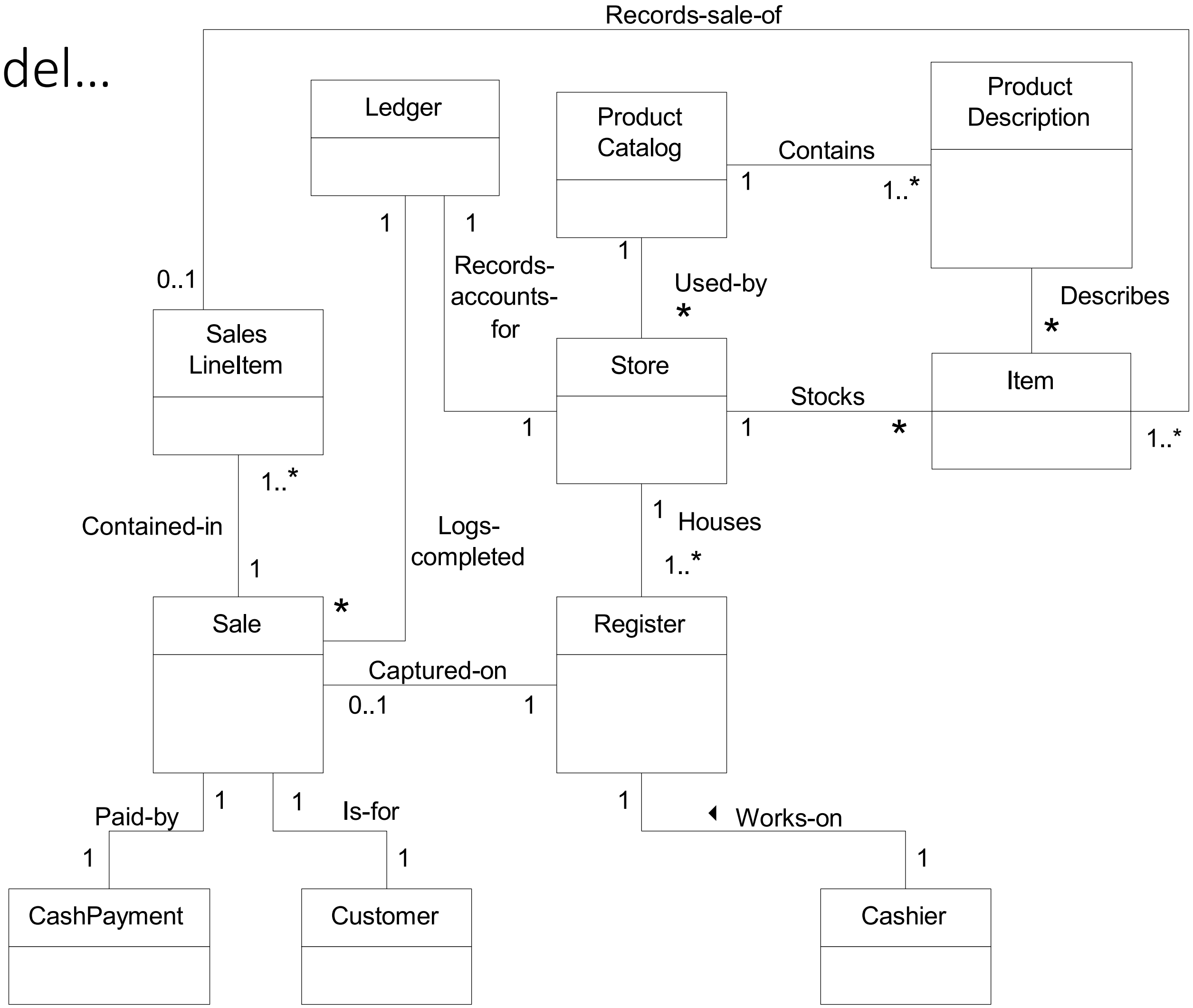
1. **Customer** arrives at a POS **checkout** with **goods** and/or **services** to purchase.
2. **Cashier** starts a new **sale**.
3. **Cashier** enters **item identifier**.
4. **System** records **sale line item** and presents **item description**, **price**, and running **total**. Price calculated from a set of price rules.

Cashier repeats steps 2-3 until indicates done.

Point of Sale System (POS) [2]

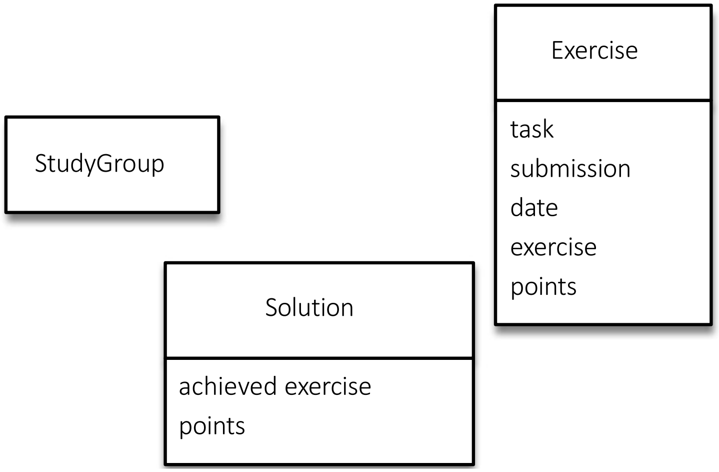
5. System presents total with taxes calculated.
6. Cashier tells Customer the total, and asks for payment.
7. Customer pays and System handles payment.
8. System logs the completed sale and sends sale and payment information to the external accounting (for accounting and commissions) and Inventory systems (to update inventory).
9. System presents receipt.
10. Customer leaves with receipt and goods (if any).

NexGen POS Domain Model…


Records-sale-of
Ledger
Product Catalog
Product Description
Contains
1
1..*
1
1
1
Records-accounts-for
Used-by
*
0..1
Sales LineItem
Store
Describes
*
Stocks
Item
1
1
*
1..*
1..*
Contained-in
1
Logs-completed
1
Houses
1..*
Sale
*
Register
Captured-on
0..1
1
1
1
1
Paid-by
Is-for
Works-on
1
1
1
CashPayment
Customer
Cashier

When should I model something as an attribute or a class?

- Rule of Thumb:
  - If we do not think of some conceptual class X as a number,
  - date or text in the real world,
  - X is probably a conceptual class,
  - not an attribute.


StudyGroup
Exercise
task
submission
date
exercise
points
Solution
achieved exercise
points

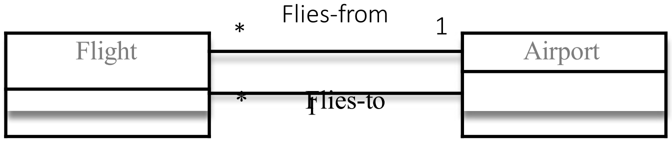
When should I model something as an attribute or a class?

Let’s assume that we develop an airline reservation system.

- Rule of Thumb:

  If we do not think of some conceptual class X as a number or text in the real world, X is probably a conceptual class, not an attribute.

- Should destination be an attribute of flight, or a conceptual class airport?
  - A destination airport is a building at a specific place, it is not just a number or some text.
  - Hence, it should be a conceptual class.
  - How about the name of the airport?



When should I add an association to the domain model?

- Rule of Thumb:
  Include associations in the domain model for which knowledge of the relationship needs to be preserved for some duration.

We are working on the conceptual model; we are not modeling associations at the software level.

When should I add an association to the domain model?

When an association is among the common associations list:

- A is a transaction related to another transaction B
- A is a line item of a transaction B
- A is a product or service for a transaction B
- A is a role related to a transaction B
- A is physical or logical part of B
- ...

When should I add an association to the domain model?

- E.g. the relation between a Sale and a SalesLineItem needs to be remembered

- However, it is not necessary to store the relation between

  - a Cashier and a ProductDescription that he looks up

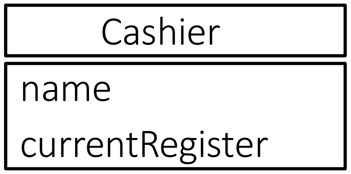
Name an association based on a ClassName- VerbPhrase-ClassName format.

The verb phrase creates a sequence that is readable and meaningful.

- Good examples:
  - Player Is-on Square
  - Sale Paid-by CashPayment
- Bad examples:
  - Sale Uses CashPayment
    (Uses is usually generic and doesn’t tell us anything.)
  - Player Has Square
    (Has is usually generic and doesn’t tell us anything.)

The attributes in a domain model should preferably be “primitive” data types (w.r.t. the domain!).

- Very common data types include: Boolean, Date, Number, Character, String, Address, Color, Phone Number, …


Cashier

name
currentRegister

- Consider modeling quantities as classes to be able to associate units

  e.g. the data type of the amount attribute of a payment should indicate the currency

The attributes in a domain model should preferably be “primitive” data types.

- Very common data types include: Boolean, Date, Number, Character, String, Address, Color, Phone Number, …

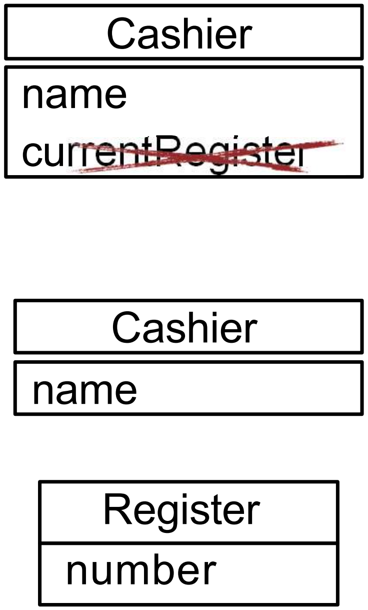
Consider modeling quantities as classes to be able to associate units

e.g. the data type of the amount attribute of a payment should indicate the currency

avoid:

- recommended:

Use associations to model dependencies between conceptual classes; do not use attributes.


Cashier

name

~~currentRegister~~

Cashier

name

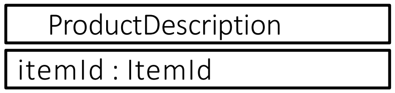
Register

number

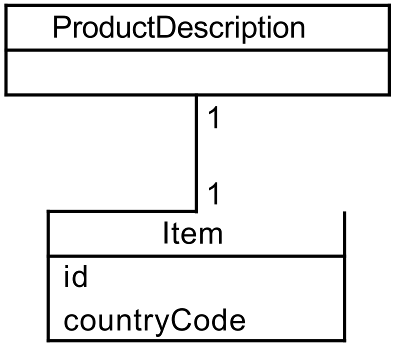
Consider defining a new data type class for something that is initially considered a string.

- If the string is composed of separate sections e.g., phone number, name of person,...
- If different operations are associated with the string e.g., social security number
- If the string has other attributes
- If the string is a quantity with a unit e.g., money has a unit for currency

avoid:


ProductDescription
itemId : ItemId

recommended:

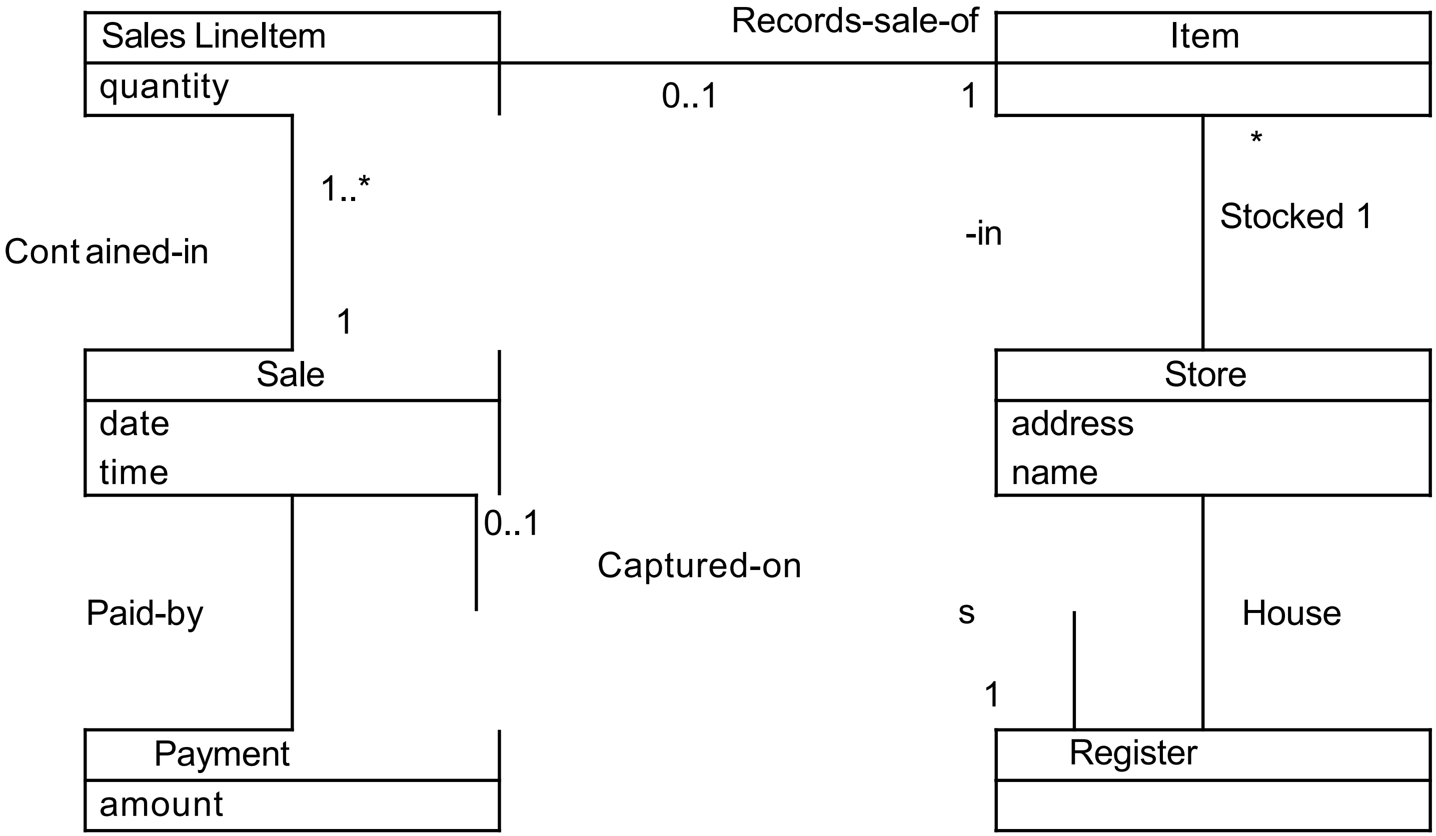

ProductDescription
1
1
Item
id
countryCode

Excerpt Of the Domain Model For the POS System

?

Which are noteworthy
domain concepts / domain objects?

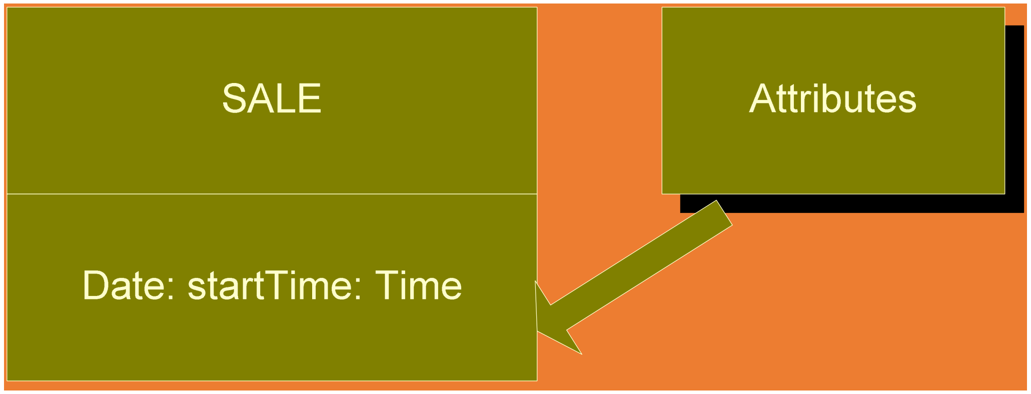
Excerpt Of the Domain Model For the POS System


Sales LineItem
quantity
Records-sale-of
0..1
1
Item
*
Stocked 1
-in
1..*
Contained-in
1
Sale
date
time
Store
address
name
0..1
Captured-on
Paid-by
s
House
1
Payment
amount
Register

Example- Order Processing system

- ATTRIBUTES
- After establishing classes based on the concepts of use case scenarios, the scenarios are examined to discover attributes
- Attributes are logical data values of an object

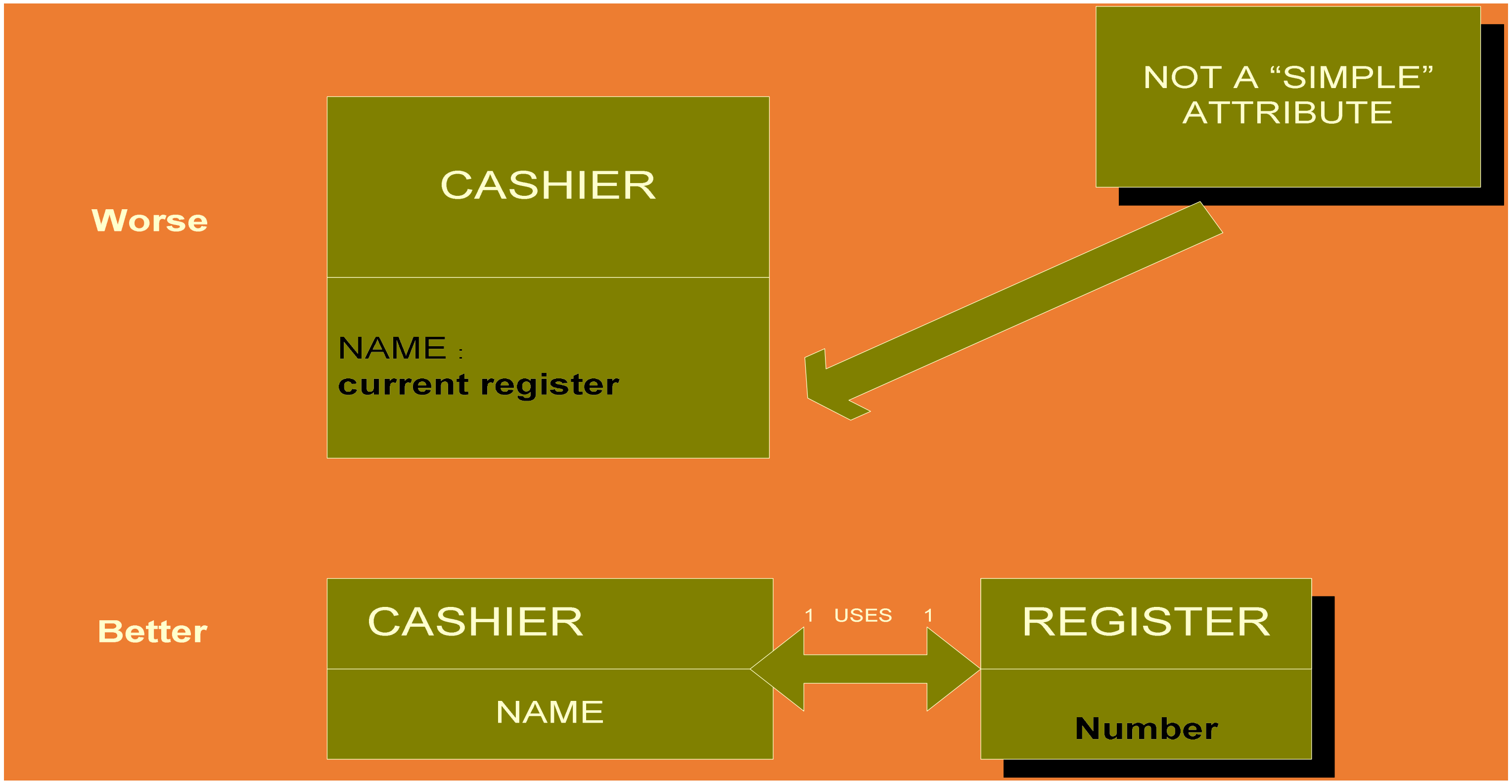
UML Attribute Notation


SALE
Date: startTime: Time
Attributes

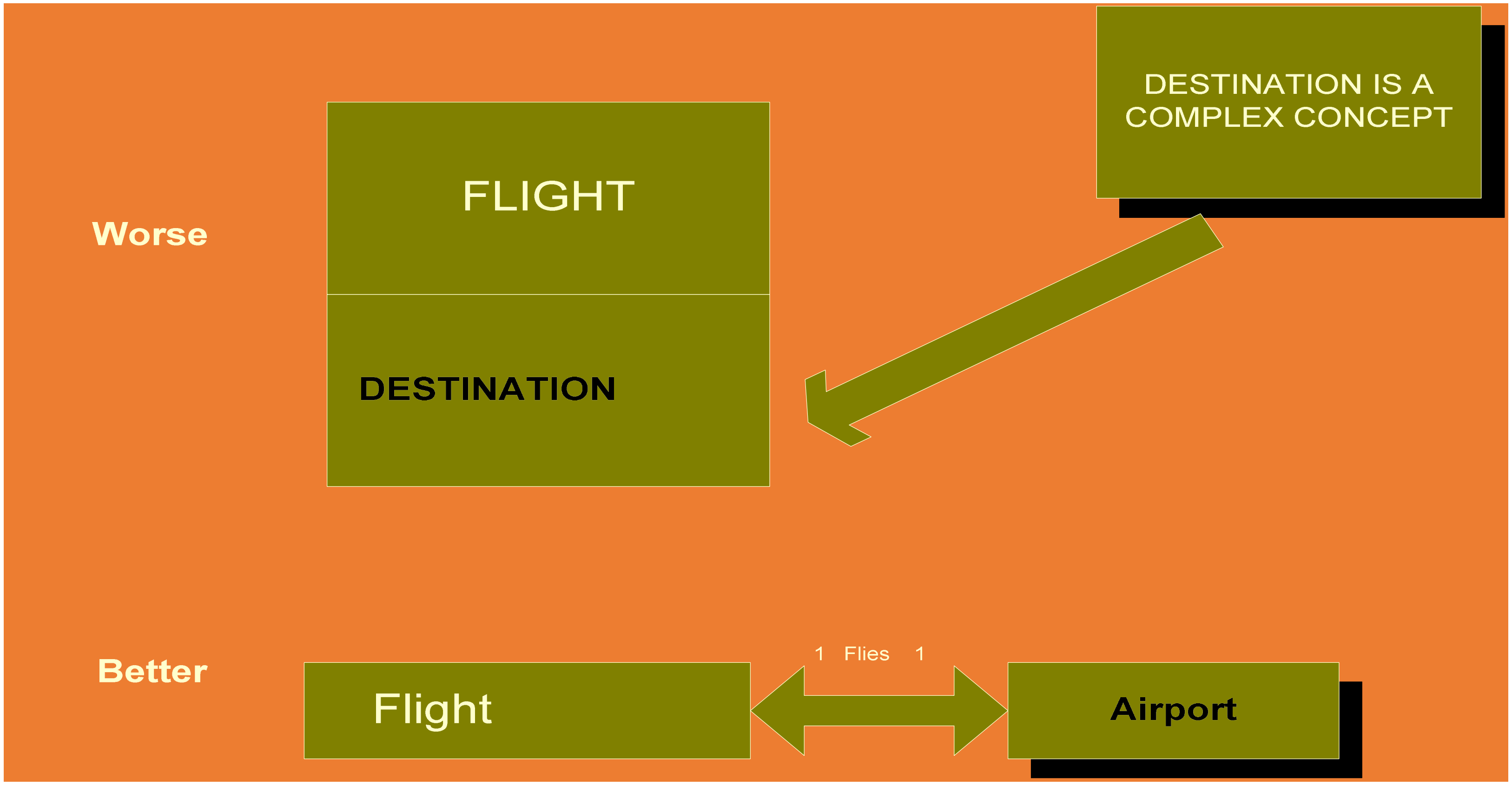
Valid Attribute Types

- Keep attributes simple
- Distinguish between conceptual and implementation perspectives
- Identify data types

Relate with associations, not attributes


NOT A "SIMPLE" ATTRIBUTE
Worse
CASHIER
NAME :
current register
Better
CASHIER
NAME
1
USES
1
REGISTER
Number

Avoid Representing Complex Domain Concepts as Attributes


Worse
FLIGHT
DESTINATION
DESTINATION IS A COMPLEX CONCEPT
Better
Flight
1 Flies 1
Airport

Non Primitive Data Type

Represent what may be considered a primitive data type (such as a number or string) as a non primitive class if:

- It is composed of separate sections.
  phone number, name of person

- There are operations usually associated with it, such as parsing or validation.
  social security number

- It has other attributes
  promotional price could have a start date and end date

Non Primitive Data Type

- It has a quantity with a unit.

payment amount has a unit of currency

- It has abstraction of one or more types with some of these qualities.

item identifier in the sales domain is a generalization of types such as Universal product code(UPC) or European Article Number(EAN)

Non Primitive Data Type

Applying these guidelines to the POS domain model yields the following analysis:

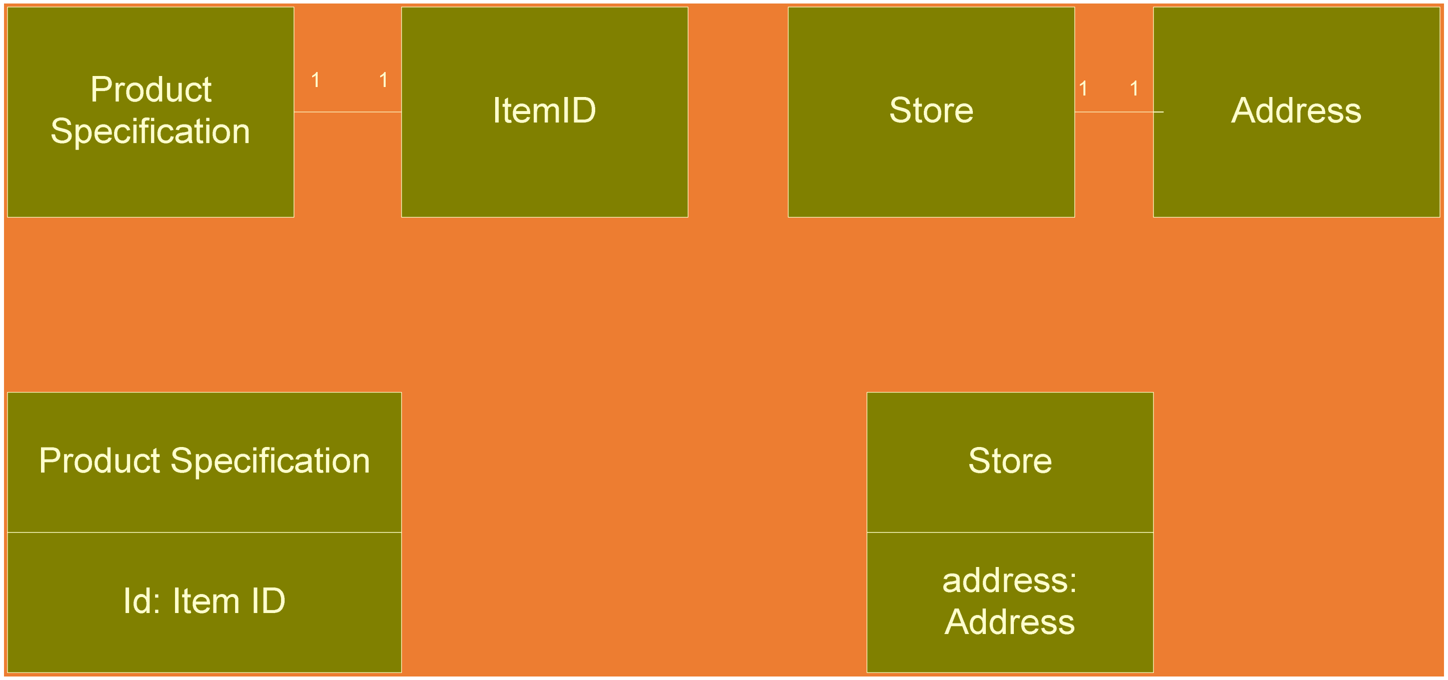
- The item identifier is an abstraction of various common coding codes schemes, including UPC-A, UPC-E, and the family of EAN schemes.
- These numeric coding schemes have subparts identifying the manufacturer, product and EAN

Non Primitive Data Type

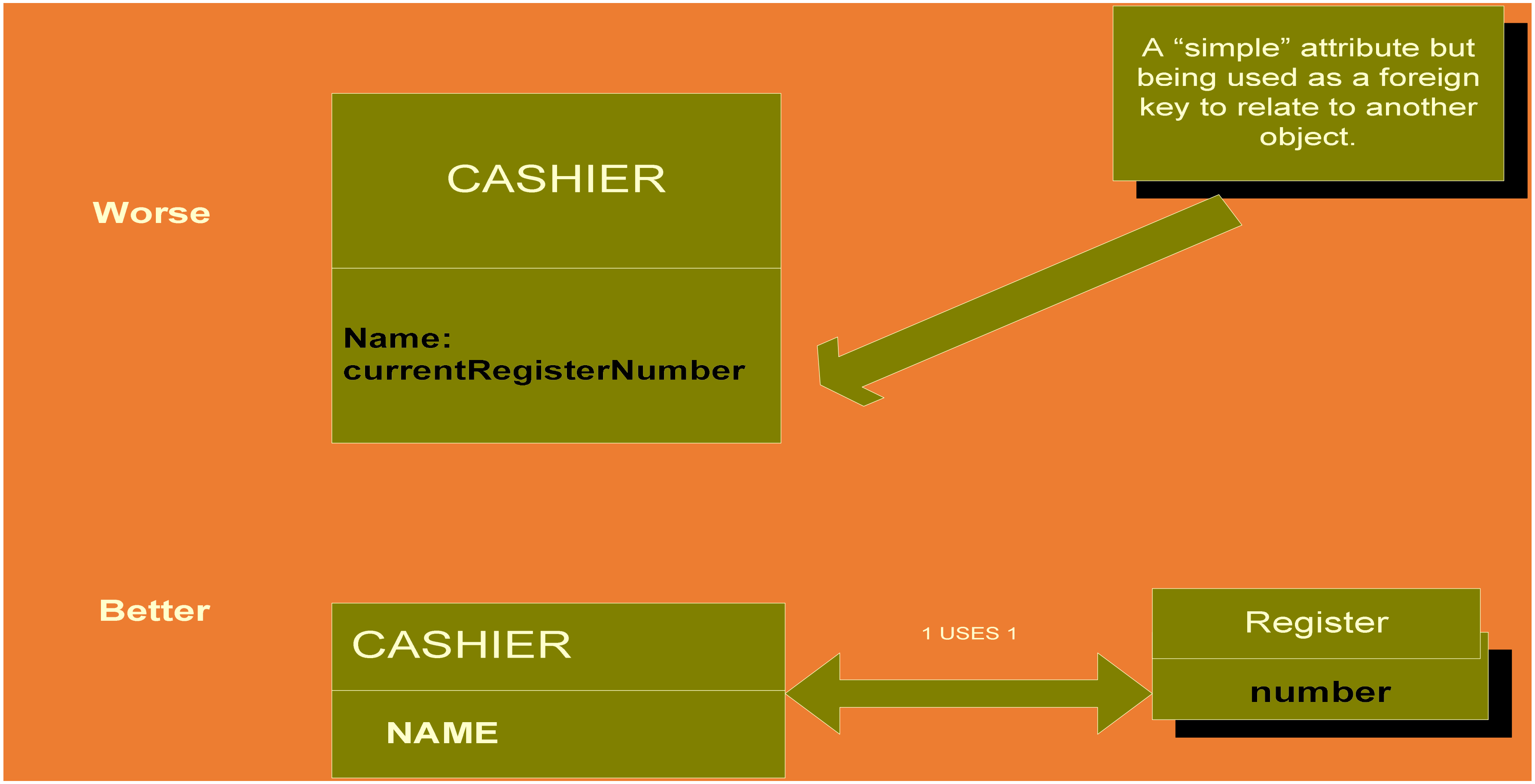
- The price and the amount attribute should be non primitive Quantity or Money classes because they are quantities in a unit of currency

- The address attribute should be a non primitive Address class because it has separate sections

If the attribute class is a data type, it may be shown in the attribute box


Product Specification
1
1
ItemID
Store
1
1
Address
Product Specification
Id: Item ID
Store
address: Address

No attributes as Foreign Key



Worse

CASHIER

Name: currentRegisterNumber

A "simple" attribute but being used as a foreign key to relate to another object.

Better

CASHIER

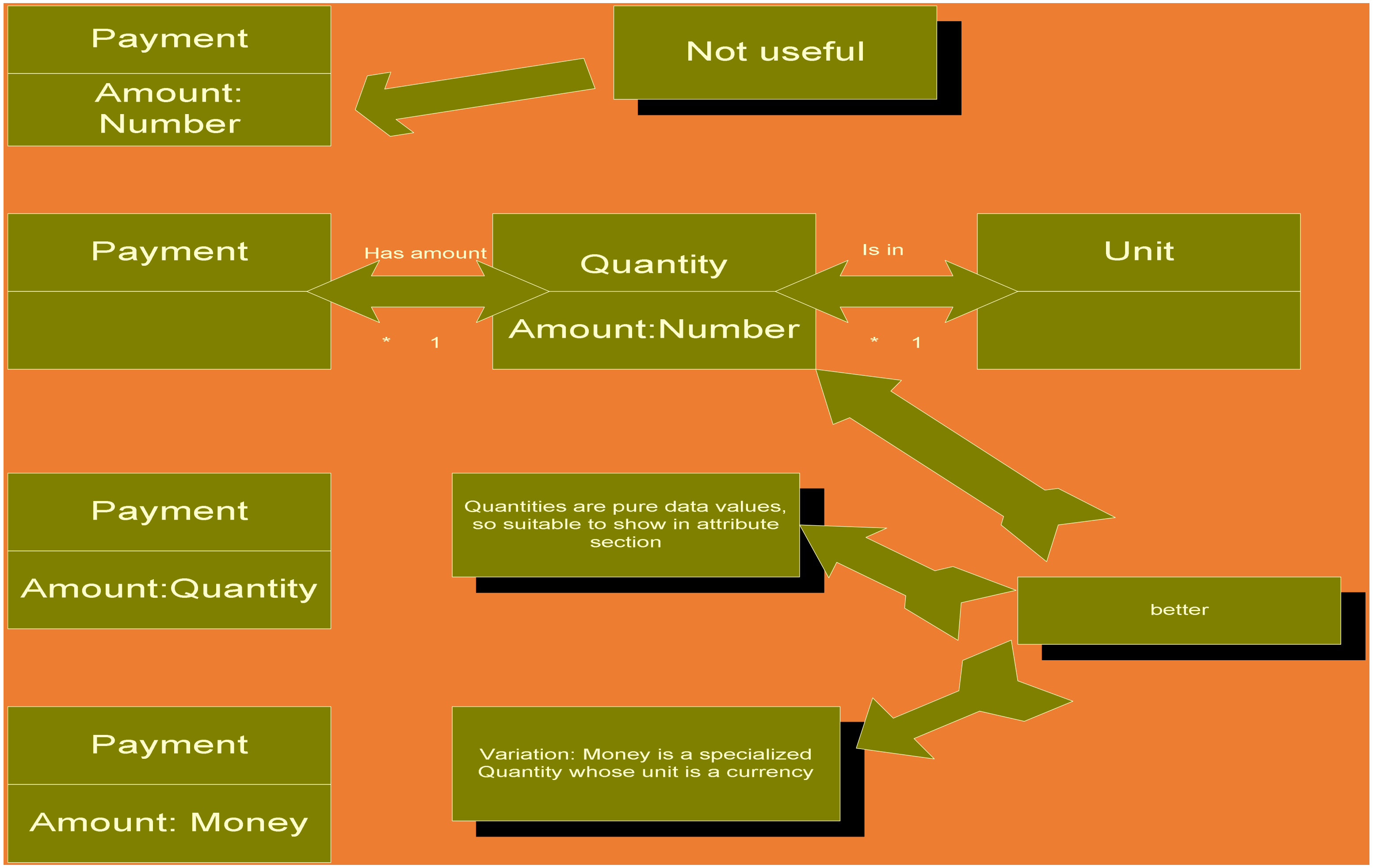
NAME

1 USES 1

Register

number

Modelling Attribute Quantities and Units



Payment

Amount: Number

Not useful

Payment

Has amount

* 1

Quantity

Amount:Number

Is in

* 1

Unit

Payment

Amount:Quantity

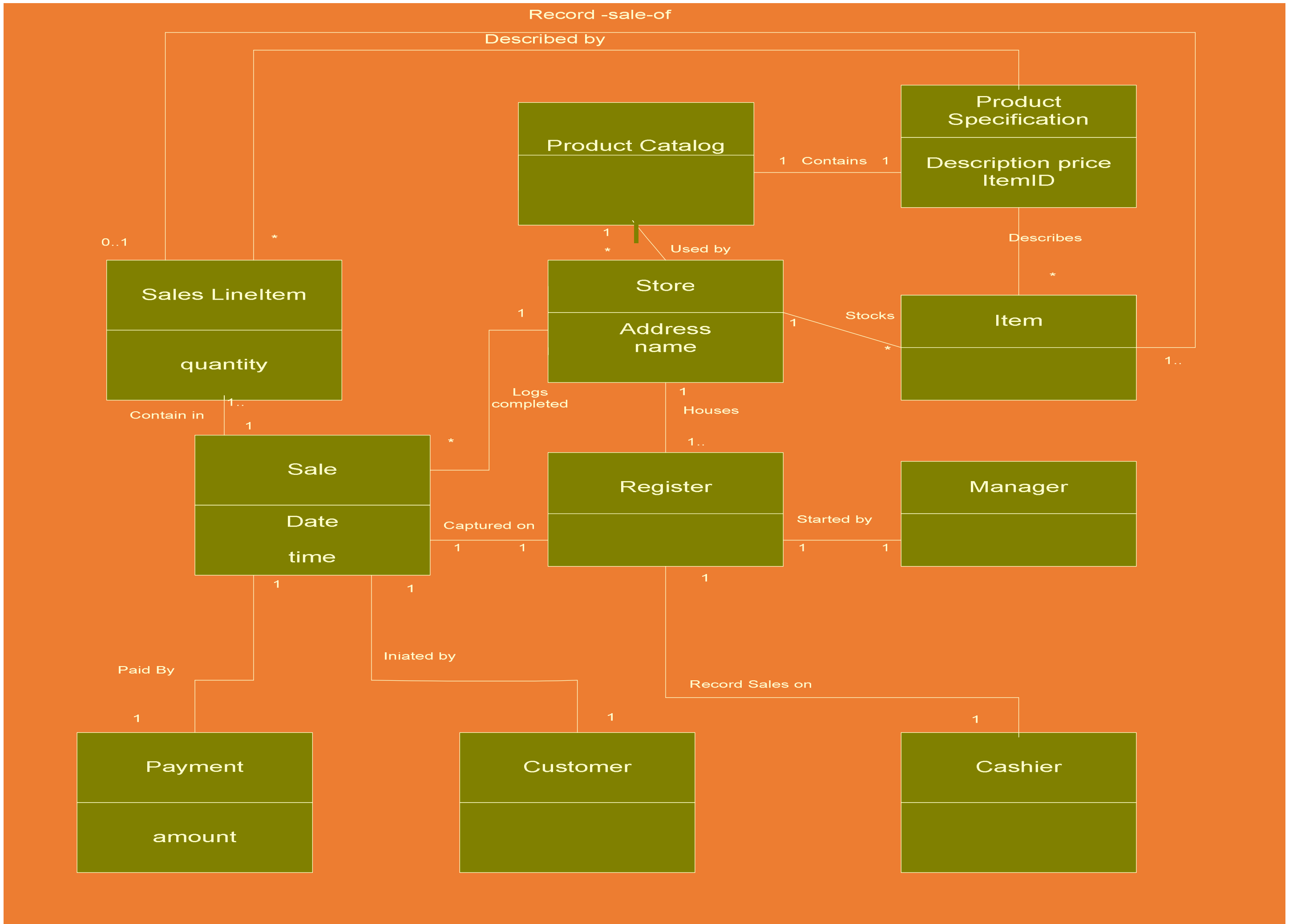
Quantities are pure data values, so suitable to show in attribute section

better

Payment

Amount: Money

Variation: Money is a specialized Quantity whose unit is a currency


Record -sale-of
Described by
Product Catalog
1 Contains 1
Product Specification
Description price ItemID
0..1
*
1
*
Used by
Describes
*
Sales LineItem
quantity
Store
Address name
1
Stocks
1
*
Item
1..
Logs completed
1
Houses
1..
Contain in
1
*
Sale
Date
time
Captured on
1
1
Register
Started by
1
1
Manager
1
1
1
Paid By
Iniated by
Record Sales on
1
1
1
Payment
amount
Customer
Cashier

A partial domain model

Domain Model Conclusion

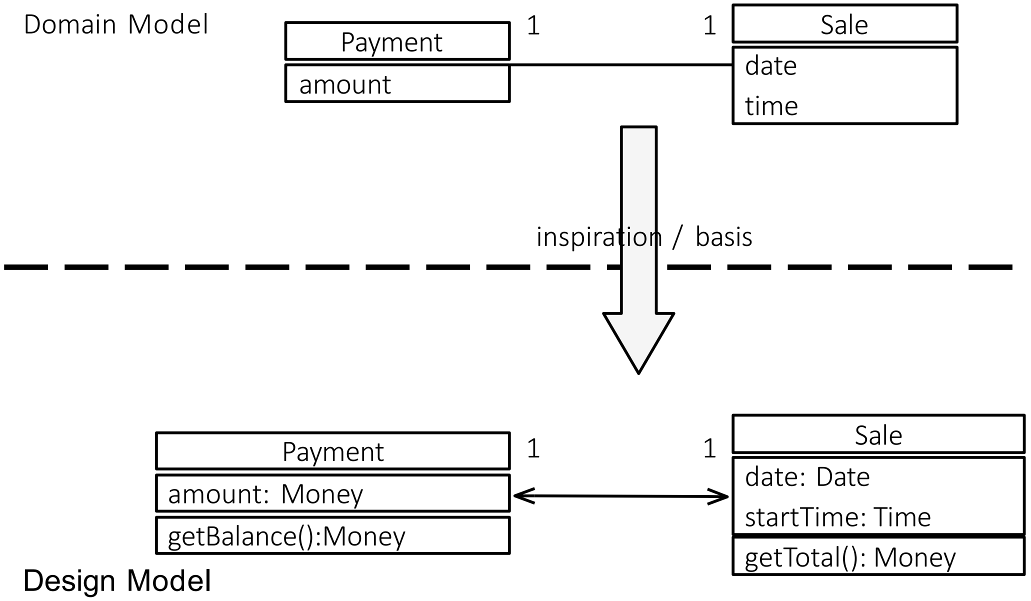
- A relatively useful model has been created for the domain of the POS application.

- A good domain model captures the essential abstractions and information required to understand the domain in context of current requirements, and aids people in understanding the domain – its concepts , terminology, and the relationships.

Domain Model and Domain Modeling

- Summary

The domain model serves as a source of inspiration for the design model.


Domain Model
Payment
amount
1
1
Sale
date
time
inspiration / basis
Payment
amount: Money
getBalance():Money
1
1
Sale
date: Date
startTime: Time
getTotal(): Money
Design Model

By using the domain model as a direct inspiration for software classes the representational gap between the domain concepts and the program is (relatively) small.

The goal of this session is to enable you to systematically carry out small(er) software projects that produce well-designed software.

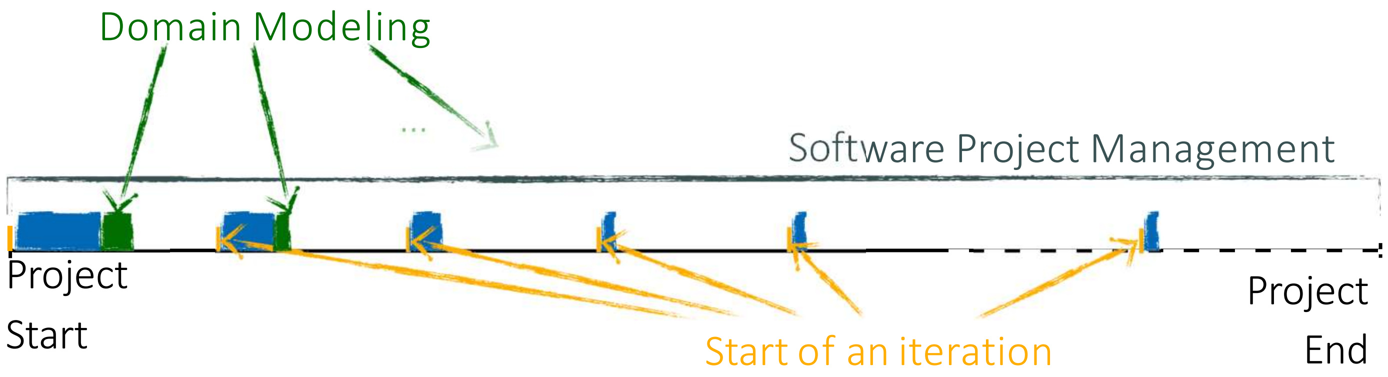
- Domain Modeling is useful to understand the ideas and concepts of the domain and their interrelationships

- A Domain Model is usually created at the beginning of a project and is a basis for the design model

- A Domain Model is created using a subset of the UML class diagram notation

🟦 Requirements Management

🟩 Domain Modeling



Domain Modeling

...

Software Project Management

Project
Start

Start of an iteration

Project
End

The goal of this lecture is to enable you to systematically carry out small(er) commercial or open-source projects.

BITS Pilani

Pilani Campus

SOFTWARE DESIGN PRINCIPLES

Class Diagram-Sub class, Association class, Association role, Aggregation & Composition

By Prof **A R Rahman**

UML
Class Diagrams: Basic Concepts

Class Diagrams

- Basic Concepts
- Type/Class
- Attribute
- Association
- Association Role
- Multiplicity
- Navigability
- Aggregation and Composition
- Association Class
- n-Ary Associations
- Generalization

Basic Concepts

- This is the main “internal” view of the system - The focus is on static structure, things about the business policy/business process that should (effectively) always be true

* A formal methods person would call these “invariants”

- The main concepts in class models are
  - Type/Class
  - Attribute
  - Association
  - Generalization

Objects

- The purpose of class modeling is to describe objects.
- An object is a concept, abstraction or thing that has meaning for a domain/application.
- Some objects have real world counterparts while others are conceptual entities.
- The choice of objects depends on judgment and the nature of problem.
- ◌All objects have identity and are distinguishable.

Classes

- An object is an instance – occurrence – of a class

- ◌̨ A class describes a group of objects with the same properties (attributes), behavior (operations), kinds of relationships, and semantics.

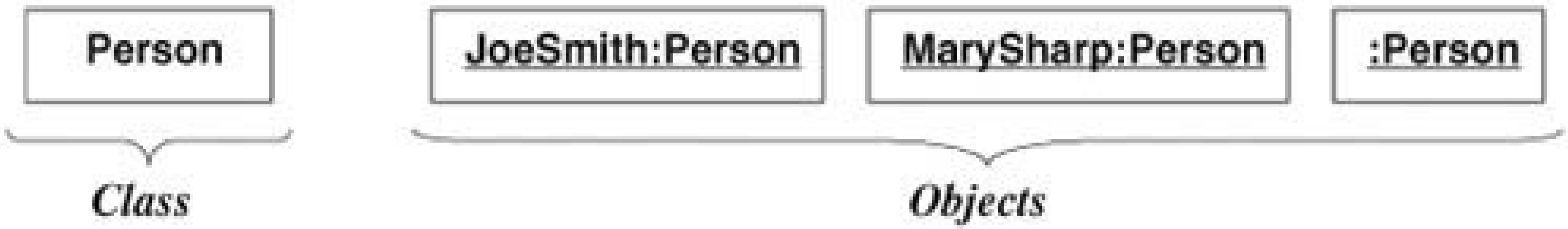
- The objects in a class share a common semantic purpose, above and beyond the requirement of common attributes and behavior.

- By grouping objects onto classes we abstract a problem.

UML representation of classes/objects:

- UML: Unified Modeling Language (OMG Standard): O.O Visual Modeling language

- Class/object representation

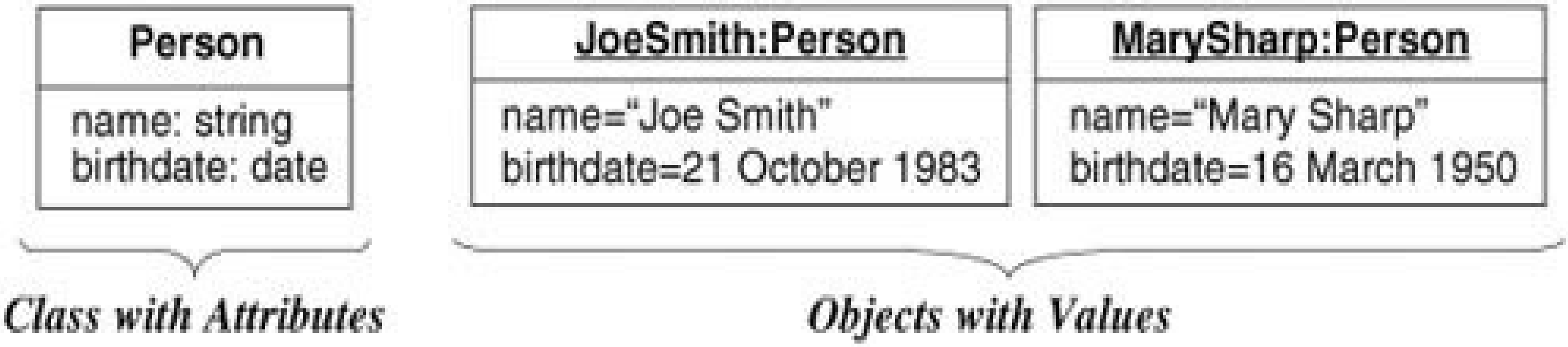

Person
JoeSmith:Person
MarySharp:Person
:Person
Class
Objects

Values and attributes

- Value : piece of data.

- Attribute: a named property of a class that describes a value held by each object of the class.
  - Attributes may be discovered by looking for adjectives or by abstracting typical values.
  - Don’t confuse values with objects:
    - An attribute should describe values, not objects.
    - Unlike objects, values lac identity

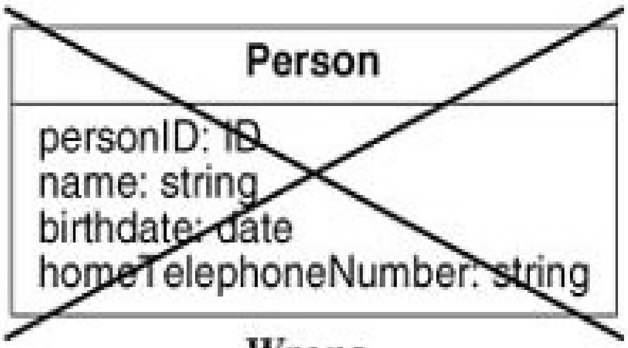
UML representation


Person
name: string
birthdate: date
JoeSmith:Person
name="Joe Smith"
birthdate=21 October 1983
MarySharp:Person
name="Mary Sharp"
birthdate=16 March 1950
Class with Attributes
Objects with Values

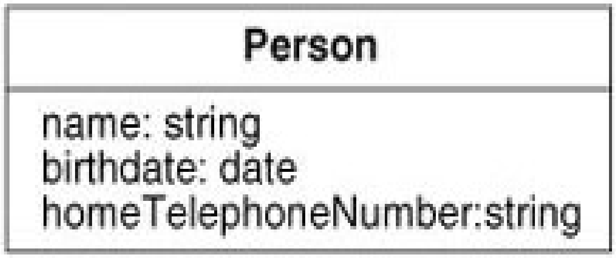
Object identifiers

- Objects identifiers are implicit.

- Objects belonging to the same and having the same attributes values may be different individual objects.


Person
personID: ID
name: string
birthdate: date
homeTelephoneNumber: string

Wrong


Person
name: string
birthdate: date
homeTelephoneNumber:string

Correct

Operations and Methods

- An operation is a function or procedure that may be applied to or by objects in a class.
- Each operation has a target object as an implicit parameter.
- All objects in a class share the same operations.
- The behavior of the operation depends on the class of its target.
- The same operation may apply to many different classes. Such an operation is POLYMORPHIC.

Operations and Methods

- A method is the implementation of an operation for a class.

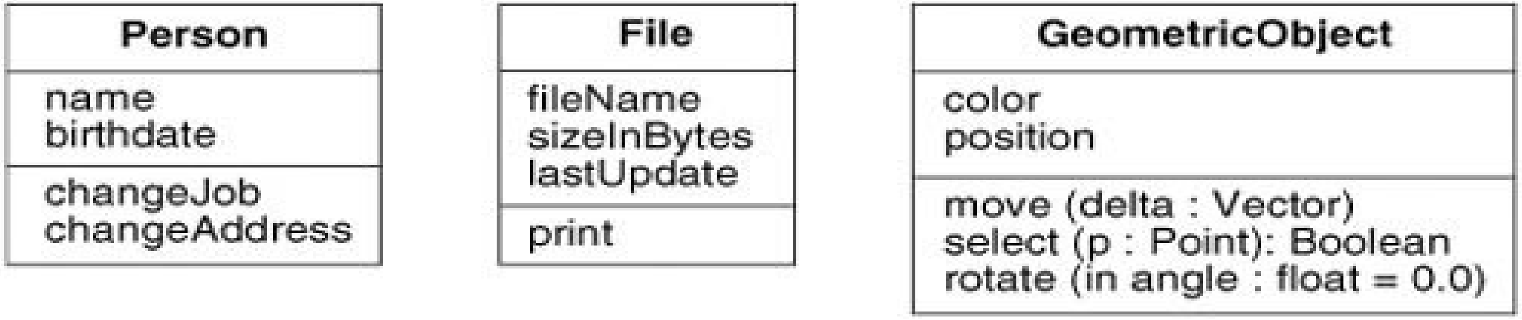
- A different piece of code may implement each method.

- An operation may have arguments in addition to its target object.

- These arguments may be placeholders for values and/or for objects.

- When an operation has methods on several classes these methods must have the same SIGNATURE: number and types of arguments, type of result value.

UML representation


Person
name
birthdate
changeJob
changeAddress
File
fileName
sizeInBytes
lastUpdate
print
GeometricObject
color
position
move (delta : Vector)
select (p : Point): Boolean
rotate (in angle : float = 0.0)

Elements of a class diagram:

class

Class Name

A class is a general concept (represented as a square box).

A class defines the structural attributes and behavioural characteristics of that concept.

Shown as a rectangle labeled with the class name.

3 Creating a Class Diagram

Here are some steps to help create a class diagram:

3.1 Identify Classes

- These are the abstract or physical "things" in our system which we wish to describe.
- Find all the nouns and noun phrases in the domain descriptions you have obtained through your analysis.
- Consider these class candidates.

3 Creating a Class Diagram

3.2 Find Associations

- Now find the verbs that join the nouns,
- e.g., The professor (noun) teaches (verb) students (noun).
- The verb in this case, defines an association between the two nouns.
- Identify the type of association.
- Use a matrix to define the associations between classes.

3 Creating a Class Diagram

3.3 Draw Rough Class Diagram

- Put classes in rectangles and draw the associations connecting the classes.

3.4 Fill in Multiplicity

- Determine the number of occurrences of one class for a single occurrence of the associated class.

3.5 Identify Attributes

Name the information details (fields) which are relevant to the application domain for each class.

3 Creating a Class Diagram

3.6 Identify Behaviours –

Specify the operations that are required for each class, (assume getter and setter methods for each attribute.)

3.7 Review your diagram and fine tune it.

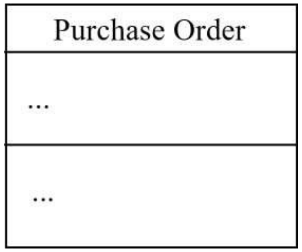
- Look for inconsistencies and errors.
- Fix them.
- Make sure you have captured everything required from the domain you are studying - that your diagram is complete.

Type/Class

- A Class is the descriptor for a set of objects with similar structure, behavior, and relationships

  - “Class” tends to be interpreted as a design and implementation concept while “Type” tends to be interpreted as an analysis concept

    - Notation(s)



Default attribute compartment

Default operation compartment

Purchase
Order
operations

responsibilities
exceptions

- Think of a type or class as representing a stencil from which the actual objects are created

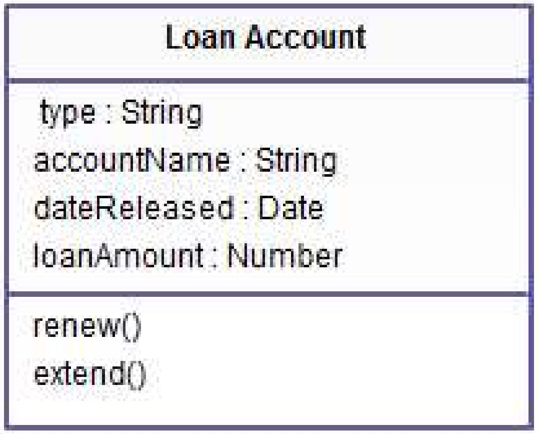
Benefits of Class Diagram

- Class Diagram Illustrates data models for even very complex information systems
- It provides an overview of how the application is structured before studying the actual code.
- This can easily reduce the maintenance time
- It helps for better understanding of general schematics of an application.
- Allows drawing detailed charts which highlights code required to be programmed
- Helpful for developers and other stakeholders.

Class Diagram

In the example, a class called “loan account” is depicted.

Classes in class diagrams are represented by boxes that are partitioned into three:


Loan Account
type : String
accountName : String
dateReleased : Date
loanAmount : Number
renew()
extend()

1. The top partition contains the name of the class.

2. The middle part contains the class’s attributes.

3. The bottom partition shows the possible operations that are associated with the class.

Type/Class (cont..)

- The following kinds of real-world things tend to end up being modeled as classes
  - Physical things
    - e.g., Airplane, Aardvark, Cyclotron
  - Important concepts
    - e.g., Account, Purchase Order
  - Roles (uses) of physical things or concepts
    - e.g., Customer, Branch Manager, Active Runway
  - Incidents (significant happenings)
    - e.g., Trouble Report, Milestone
  - Characteristics
    - e.g., Airplane-type, Computer-model, Recipe
  - Associations
    - * e.g., Enrollment, License

- Include each of the actors from the use cases on the class diagram
  - Suggestion: stereotype them as «actor»

Attribute

What is known about each object of this class type. Typical characteristics.

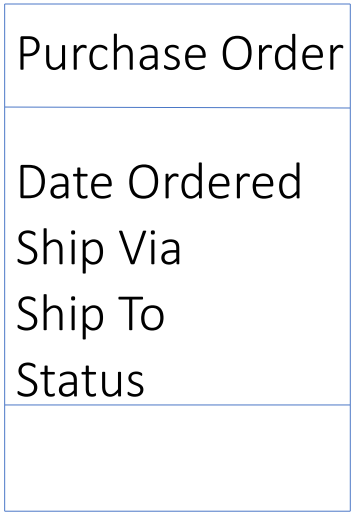
class 1

-attribute 1
-attribute 2
-attribute 3

Attribute

- An Attribute is a single, named fragment of the persistent state

- Notation



Purchase Order

Date Ordered
Ship Via
Ship To
Status

operation

- What objects of this class type do.
- Typical actions that are relevant to the current analysis.

class 1

-attribute 1

-attribute 2

-attribute 3

+operation 1()

+operation 2()

+operation 3()

Links and Association concepts

- A link is a physical or conceptual connection among objects.
- Most links relate two objects, but some links relate three or more objects.
- A link is defined as a tuple, that is a list of objects.
- A link is an instance of an association.
- An association is a description of a group of links with common structure and semantics.

Links and Association concepts

- Association is denoted by a line. Its name is optional if the model is unambiguous.

- Associations are inherently bi-directional.

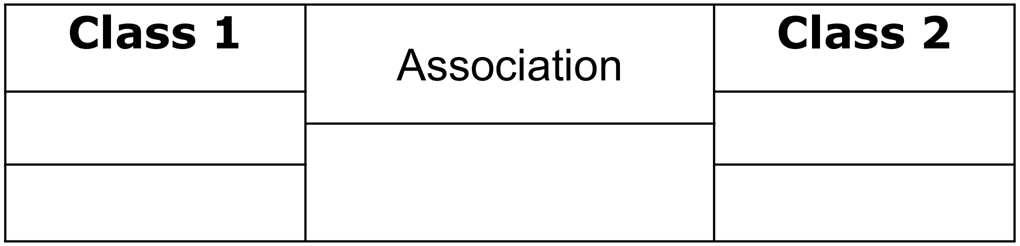
- The association name is usually read in a particular direction but the binary association may be traversed in either direction.

- ◌̦ A reference is an attribute in one object that refers to another object.

Association

A (semantic) relationship between classes.

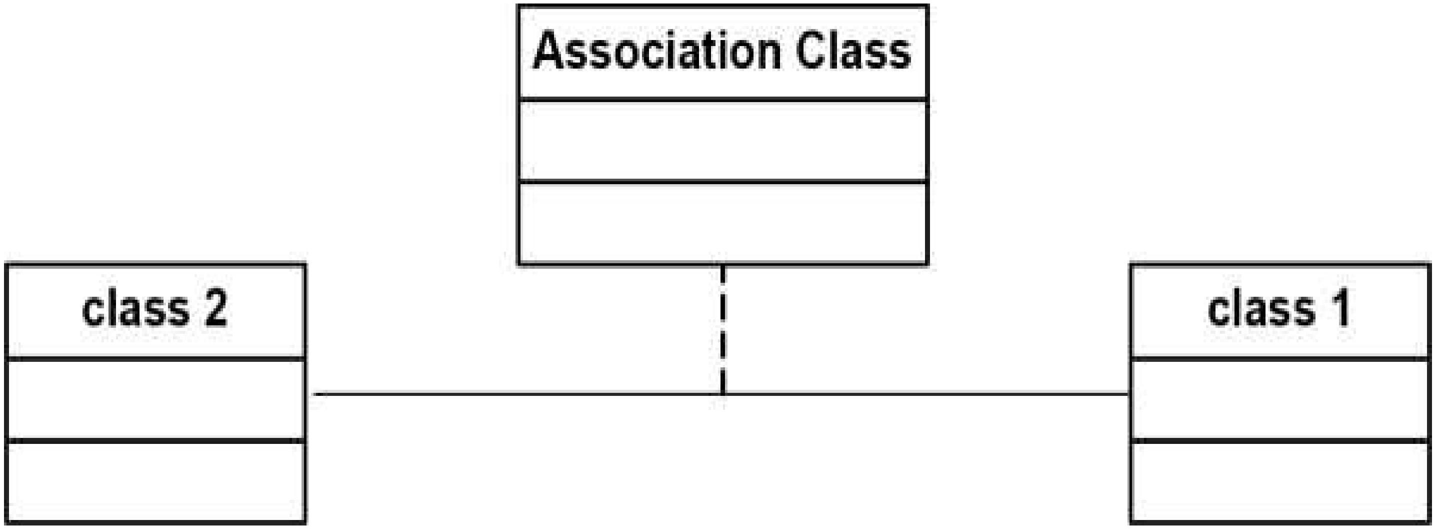
A line that joins two classes.


Class 1
Association
Class 2

Association

Can be applied to binary or n-ary associations.

Used to capture the attributes of an association.


Association Class
class 2
class 1

Association

- An Association exists when some number of objects are connected (linked) in some significant manner and it is necessary to remember that connection

  - In set theory terms, each class corresponds to a set of things, and an association corresponds to a mapping between members of these sets

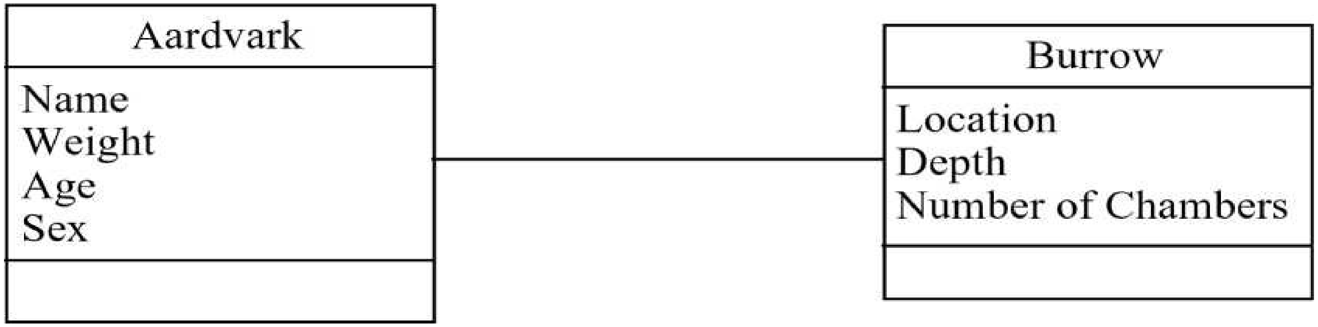
- Associations permit dynamics of the form

Given some member of set A, which member or members of set B is it linked with?

Given an aardvark, which burrow does it live in?

Given a burrow, which aardvark lives in it?

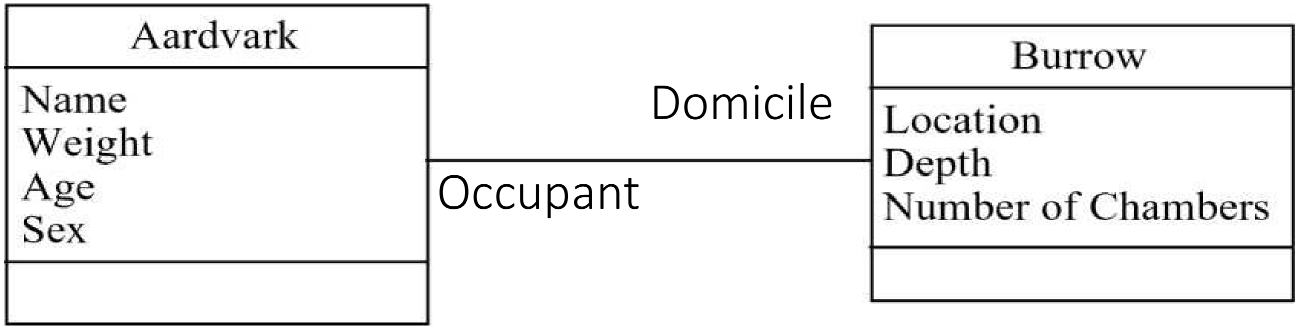
Basic notation


Aardvark
Name
Weight
Age
Sex
Burrow
Location
Depth
Number of Chambers

Association Role

- An Association Role indicates the role (purpose) that the class plays in the association

- Notation


Aardvark
Name
Weight
Age
Sex
Domicile
Occupant
Burrow
Location
Depth
Number of Chambers

- Roles can be explicitly named, but they don’t have to be - If not explicitly named, the role “defaults” to the name of the target class

Multiplicity

- Describes the rules about the number of objects in each class that may or may not be linked

- "Anchor point analysis" [Flavin81] is a useful tactic to help you determine multiplicity. It is based on the question

"Assuming that I am an object of this class, what is the minimum and maximum number of objects in that class can I be linked with? "

The question is repeated from the point of view ("anchored at") each of the participating classes

- Example

If I were an Aardvark, what is the minimum and maximum number of Burrows I could live in?

If I were a Burrow, what is the minimum and maximum number of Aardvarks that could live in me?

Multiplicity

- It specifies the number of instances of one class that may relate to a single instance of the associated class.
- UML diagrams explicitly list multiplicity at the end of association lines.
- Intervals are used to express multiplicity:
  - 1 (exactly one)
  - 0..1 (zero or one)
  - 1..* (one or more)
  - 0..* (zero or more)
  - 3..5 (three to five inclusive)

Notation

Lower bound	Upper bound	Notation
0	1	0..1
0	Many	*
1	1	1
1	Many	1..*

[Flavin81]

Matt Flavin, Fundamental Concepts of Information Modeling, Yourdon Press, 1981

Multiplicity (cont..)

- The following are examples of the various forms of multiplicity

On-Duty Crew IS ASSIGNED TO Ship at Sea (1:1)

Operating Power Supply POWERS Amplifier (0.. 1:1)

Batch of Chocolate IS CONTAINED IN Vat (0..1:0..1)

Slum Lord OWNS Apartment (1:1..*)

Multiplicity (cont..)

- The following are examples of the various forms of multiplicity

Python Variable HAS DEFINED Data Type (1: *)

Secretary IS ASSIGNED TO Manager (0..1:*)

Student IS ENROLLED IN Course (l..*:l..*)

Clubmember IS MEMBER OF Committee (1..*:*)

Traveler HAS RESERVATION ON Scheduled Flight (*:*)

Multiplicity (cont..)

- Multiplicity may be even further constrained by the business policy/business process

Commercial Pilot HOLDS RATING IN Airplane-Type (*: 0..2)

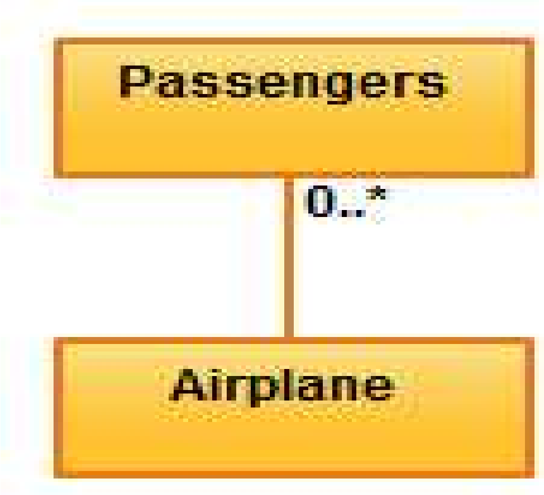
- Document the policy-constrained limits in the model

Multiplicity (cont..)

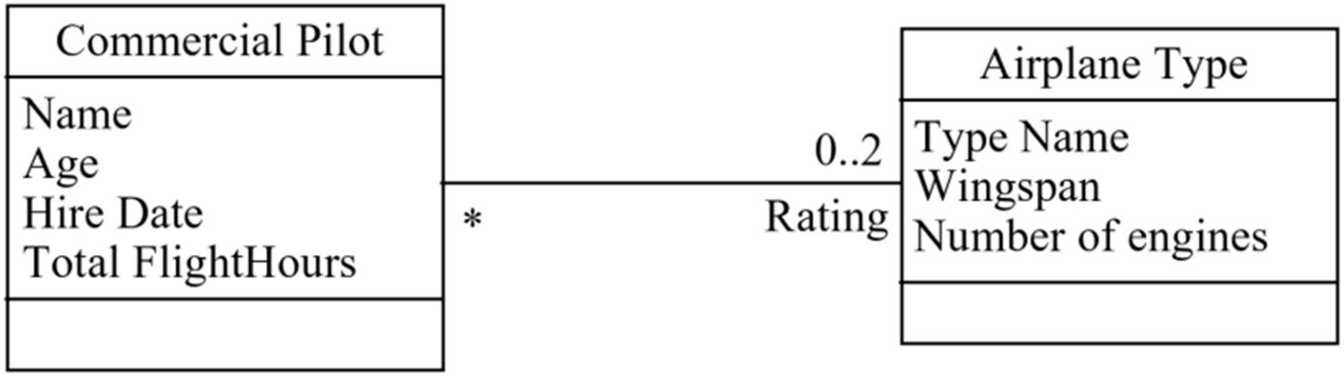
is the active logical association when the cardinality of a class in relation to another is being depicted.

For example, one fleet may include multiple airplanes, while one commercial airplane may contain zero to many passengers.

The notation 0..* in the diagram means “zero to many”.



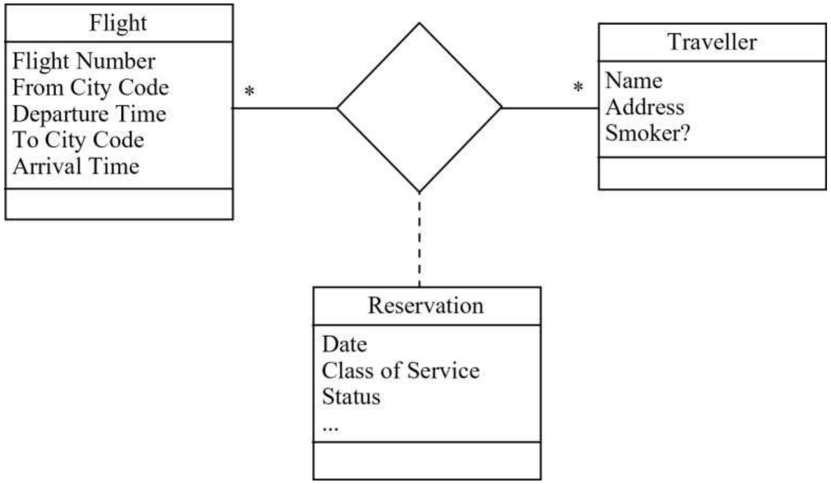
Multiplicity (cont..)


Commercial Pilot
Name
Age
Hire Date
Total FlightHours
*
0..2
Rating
Airplane Type
Type Name
Wingspan
Number of engines

Association Class

- Associations may take on the characteristics of a class

- Associations may have attributes
- Associations may have important states and actions


Flight
Flight Number
From City Code
Departure Time
To City Code
Arrival Time
*
*
Traveller
Name
Address
Smoker?
Reservation
Date
Class of Service
Status
...

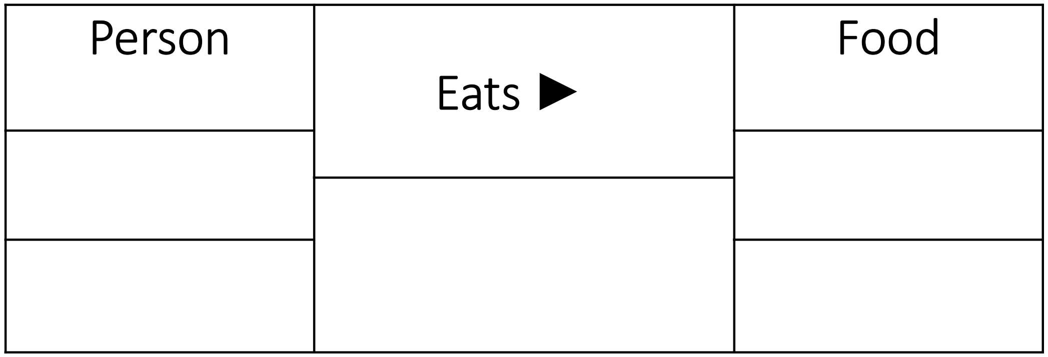
- Notation

- Even though in the notation it appears as an association and a class separately, semantically it is a single element in the model

Binary Associations

Simple association between two classes.

A solid triangle with the association name indicates the direction in which the association is meant to be read.



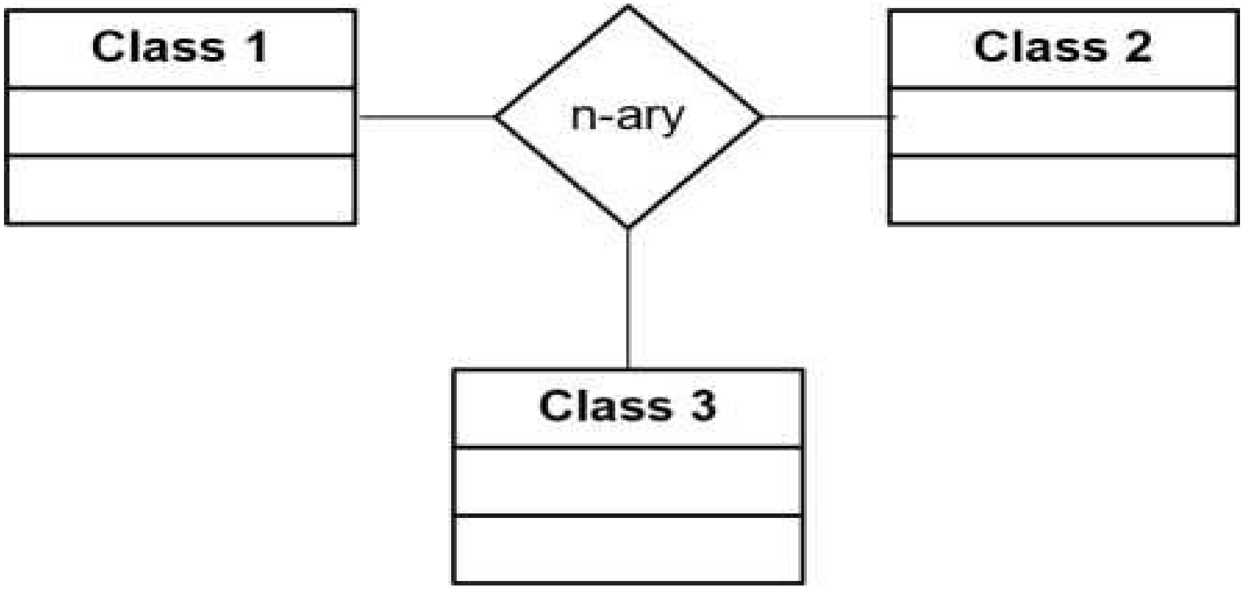
Person

Eats ▶

Food

n-ary Associations

n-ary Association expresses an association between multiple classes

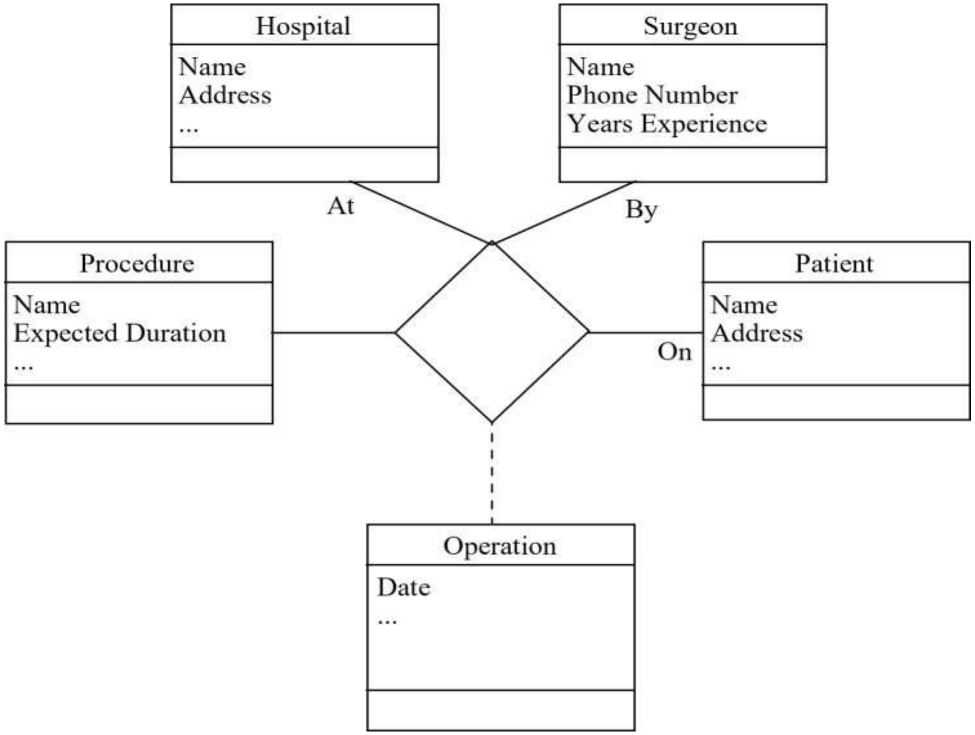

Class 1
n-ary
Class 2
Class 3

n-Ary Associations

- So far, we have only seen associations between two classes

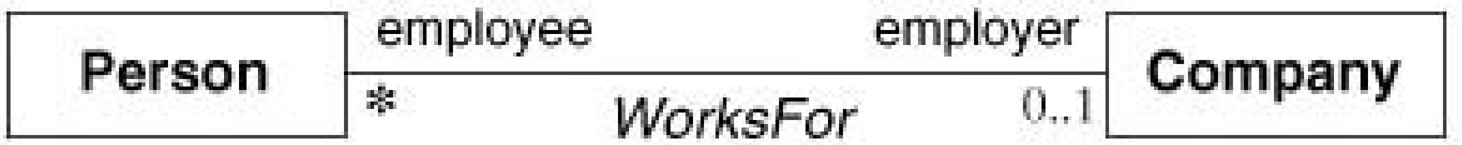
  - This is the usual case, but some associations involve more than two classes

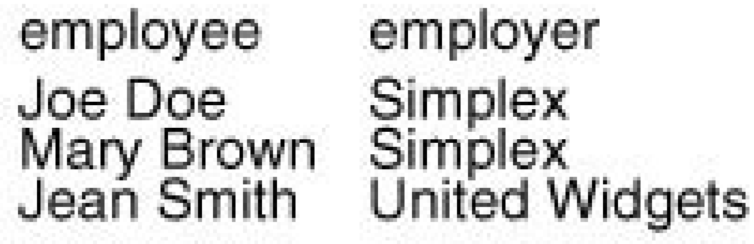
- Notation


Hospital
Name
Address
...
Surgeon
Name
Phone Number
Years Experience
At
By
Procedure
Name
Expected Duration
...
Patient
Name
Address
...
On
Operation
Date
...

Association Ends

- Associations have ends. They are called ‘Association Ends’.
- They may have names (which often appear in problem descriptions).

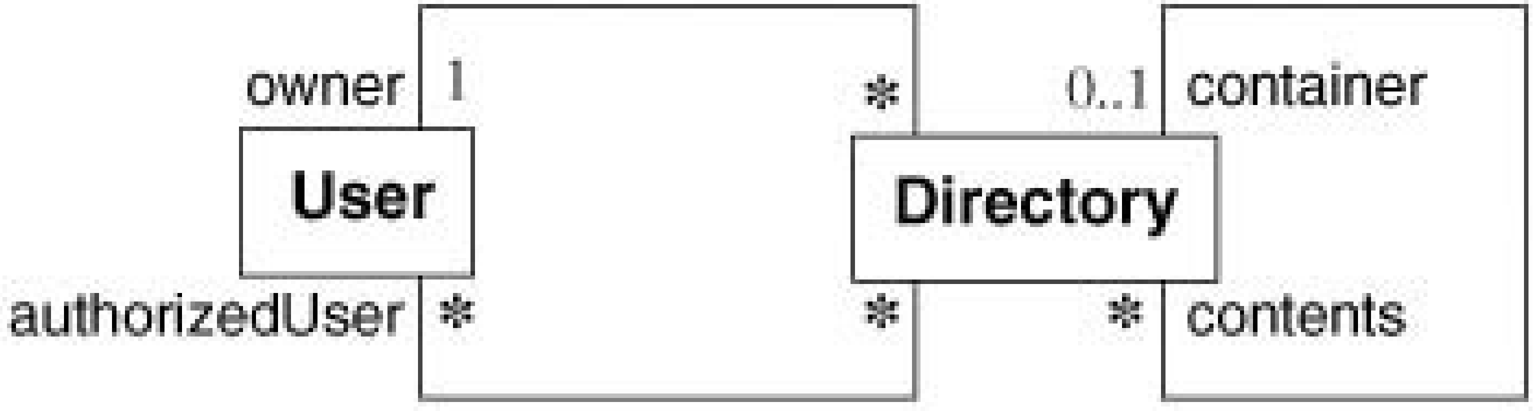

Person
employee
*
WorksFor
employer
0..1
Company



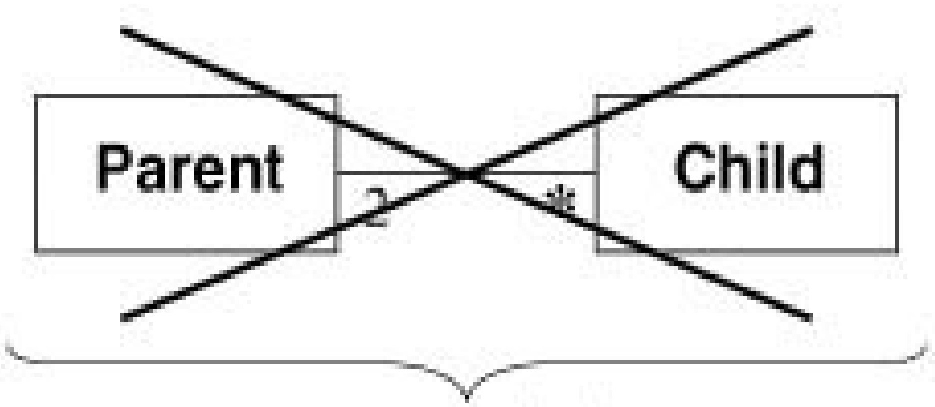
employee	employer
Joe Doe	Simplex
Mary Brown	Simplex
Jean Smith	United Widgets

Association Ends

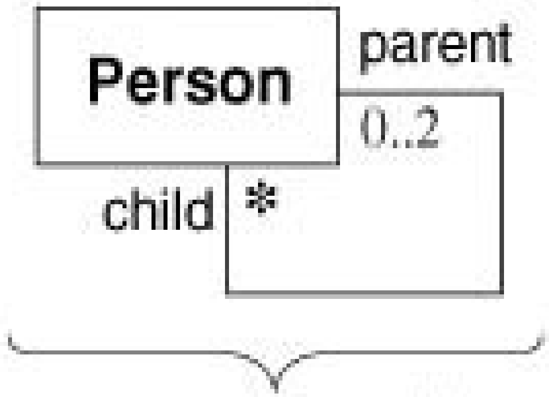
- Use of association end names is optional.
- But association end names are useful for traversing associations.
- ꙰Association end names are necessary for associations between objects of the same class.


owner
1
User
authorizedUser
*
*
*
0..1
container
Directory
*
contents

Example of association ends use


Parent
2
*
Child

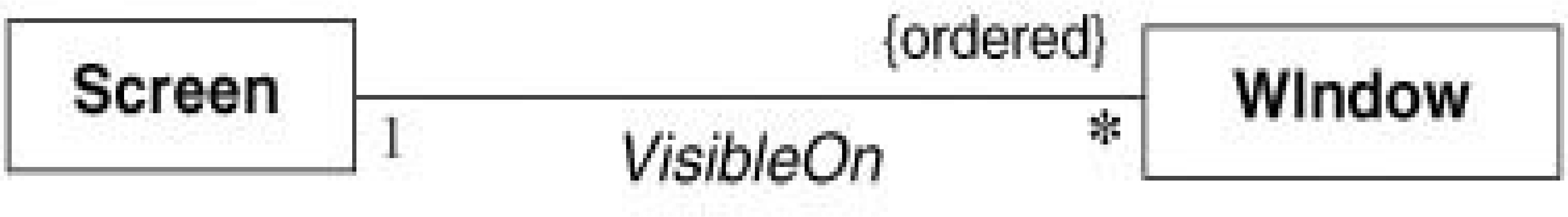
Wrong model


Person
parent
0..2
child
*

Correct model

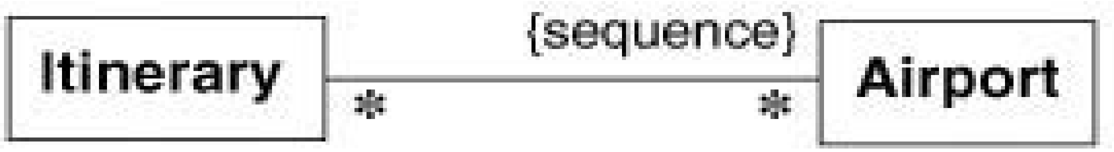
Association: ordering, bag, sequence

- On a ‘many” association end, sometimes, it is required that objects have an explicit order.
- In this case the ordering is an inherent part of the association
- Example:


Screen
{ordered}
Window
1
VisibleOn
*

Association: ordering, bag, sequence

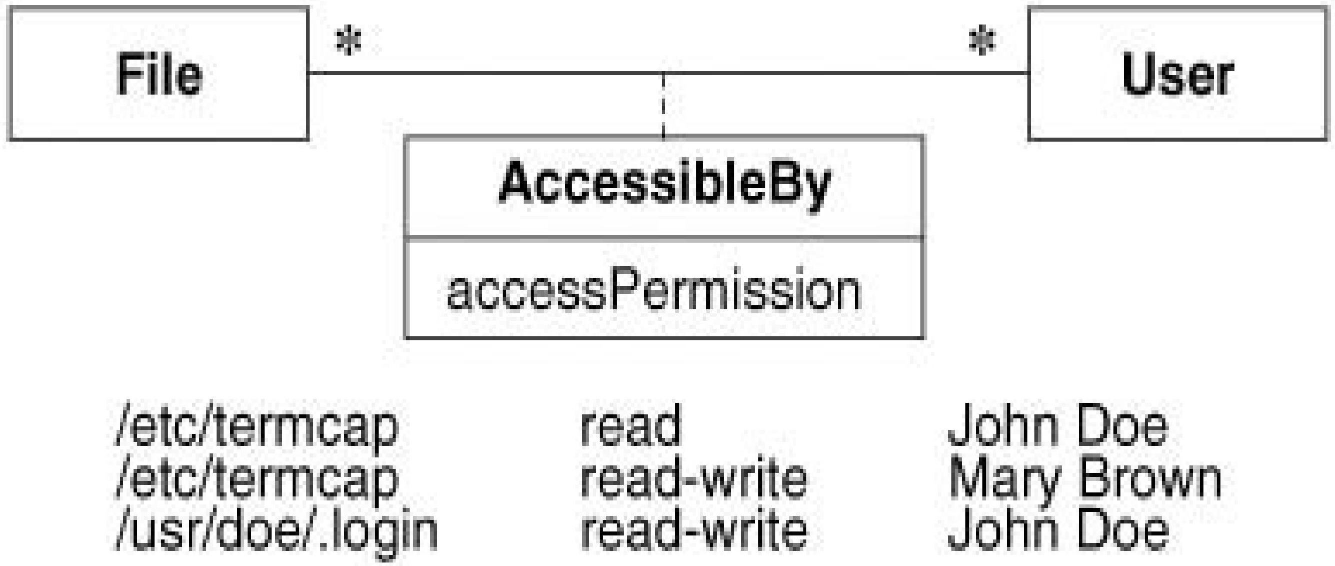
- Ordinary a binary association has at most one link for a pair of objects
- However we can permit multiple links for a pair of objects by annotating an association end with {bag} or {sequence}
- A **bag** is a collection of elements with duplicates allowed.
- A **sequence** is an ordered collection of elements with duplicates allowed



Association class

- UML offers the ability to describe links of association with attributes like any class.

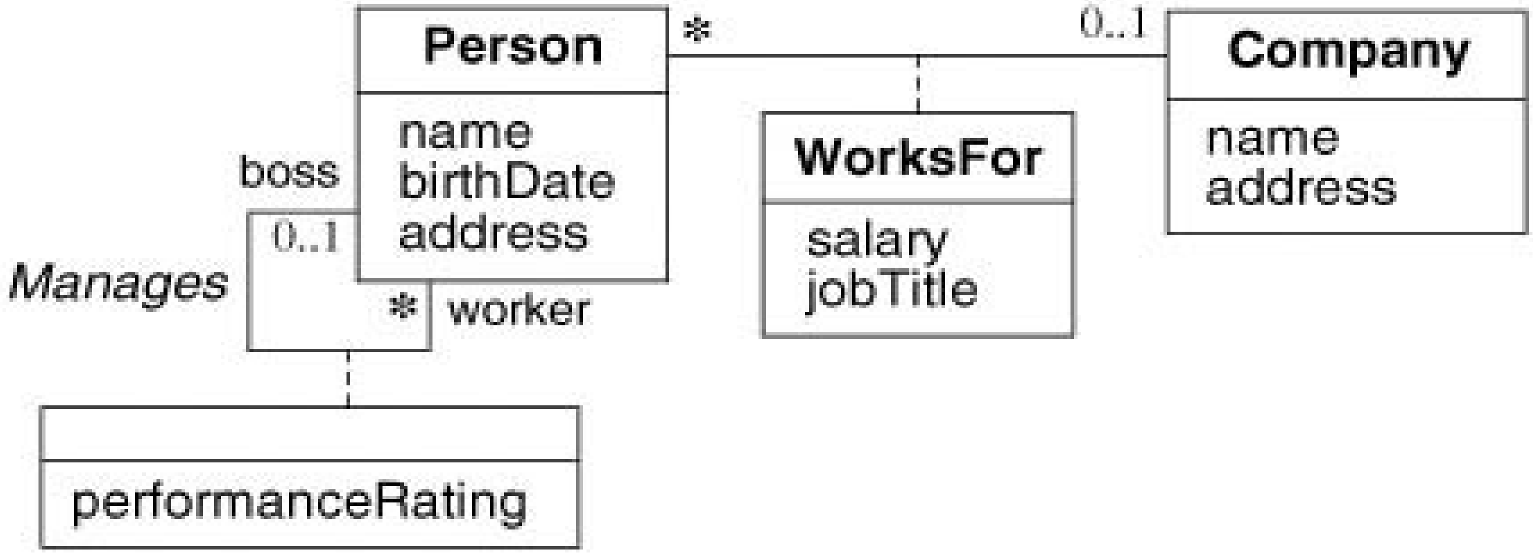
- An association class is an association that is also a class.


File
*
*
User
AccessibleBy
accessPermission

/etc/termcap	read	John Doe
/etc/termcap	read-write	Mary Brown
/usr/doe/.login	read-write	John Doe

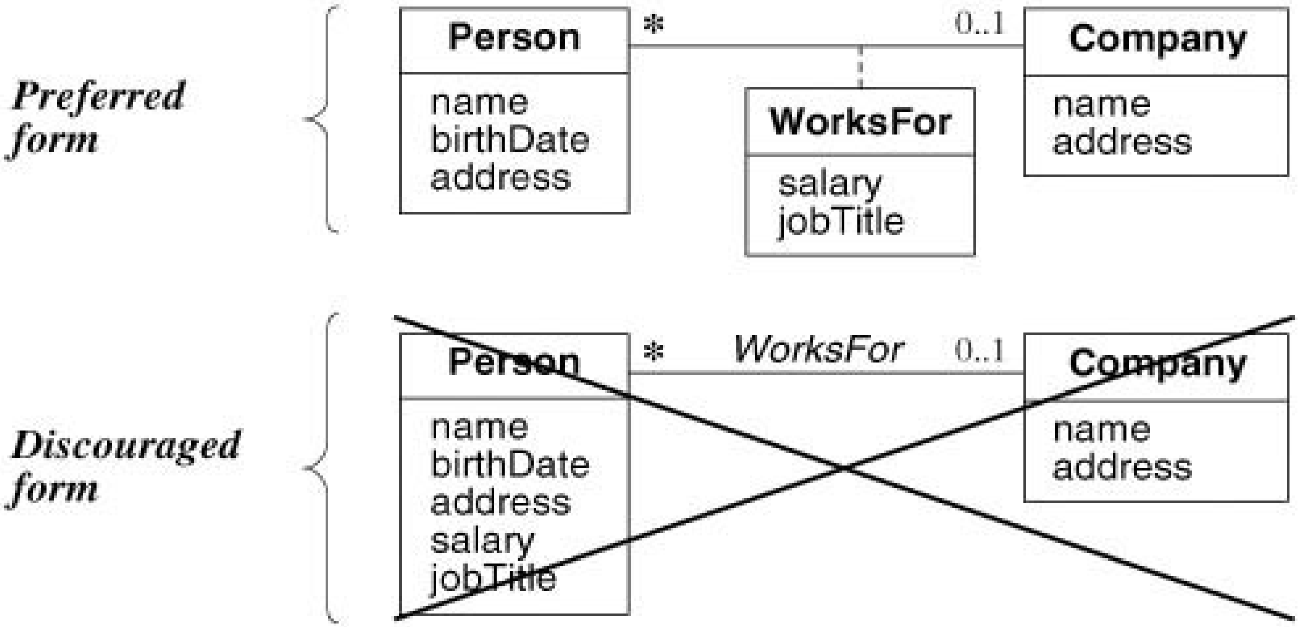
Association class

- Example:


Person
name
birthDate
address
*
0..1
Company
name
address
WorksFor
salary
jobTitle
boss
0..1
Manages
*
worker
performanceRating

Association class

- Example



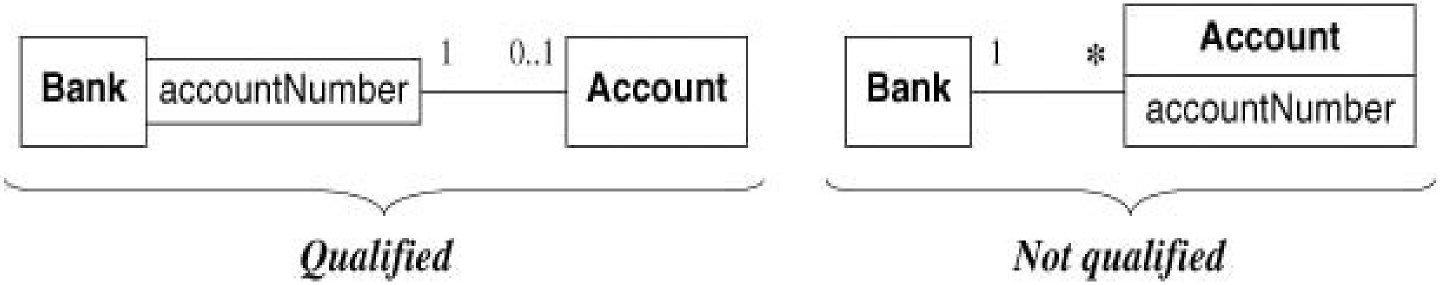
Preferred form
Person
name
birthDate
address
*
0..1
Company
name
address
WorksFor
salary
jobTitle
Discouraged form
Person
name
birthDate
address
salary
jobTitle
*
WorksFor
0..1
Company
name
address

Qualified Association

- A qualified association is an association in which an attribute called Qualifier disambiguates the objects for a 'many' association' end.

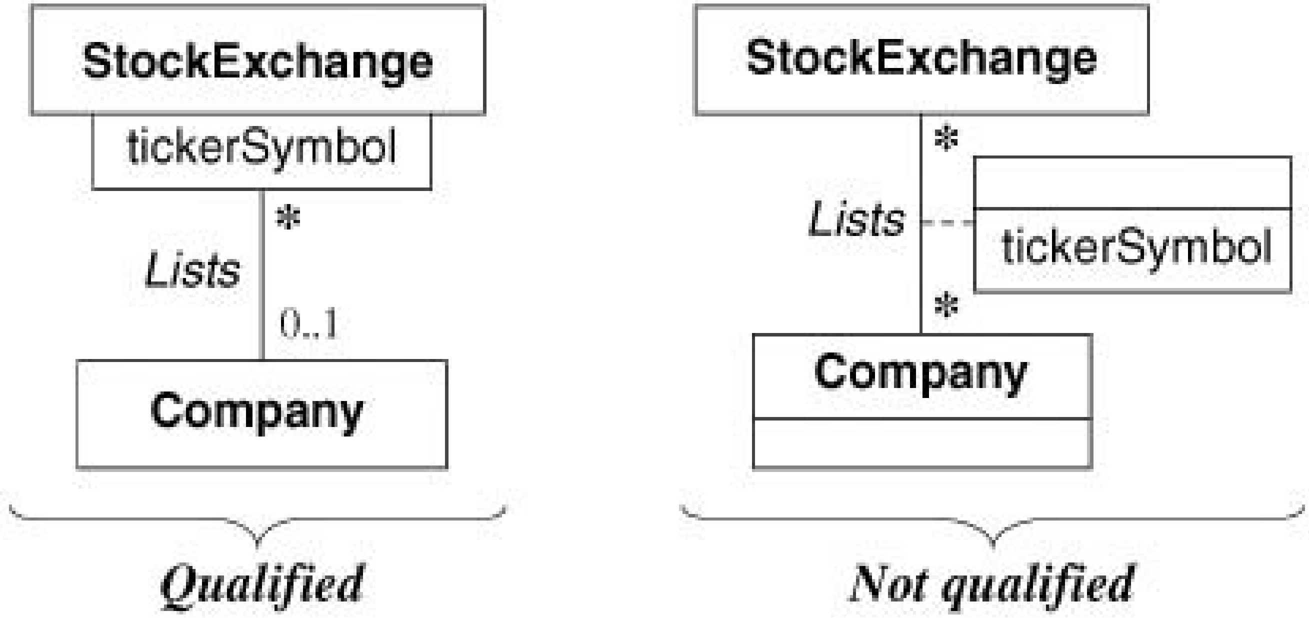
- A qualifier selects among the target objects, reducing the effective multiplicity from 'many' to 'one'.

- Both below models are acceptable but the qualified model adds information.


Bank
accountNumber
1
0..1
Account
Qualified
Bank
1
*
Account
accountNumber
Not qualified

Qualified Association

- Example:

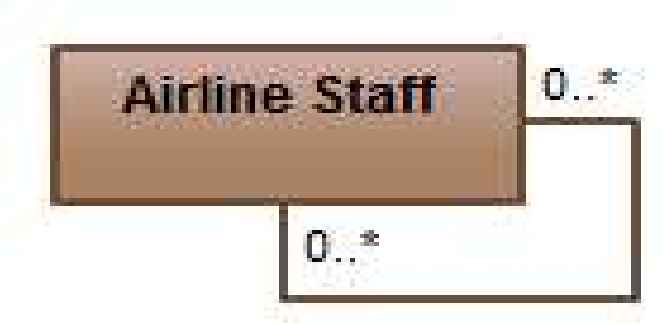

StockExchange
tickerSymbol
*
Lists
0..1
Company
Qualified
StockExchange
*
Lists
tickerSymbol
*
Company
Not qualified

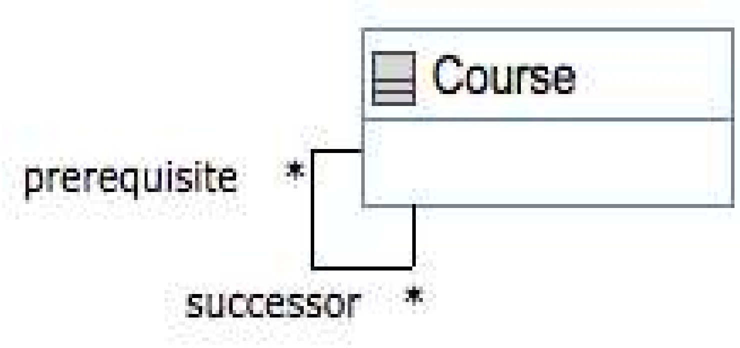
Reflexive Association

- This occurs when a class may have multiple functions or responsibilities.

- For example, a staff member working in an airport may be a pilot, aviation engineer, a ticket dispatcher, a guard, or a maintenance crew member.

- If the maintenance crew member is managed by the aviation engineer there could be a managed by relationship in two instances of the same class.

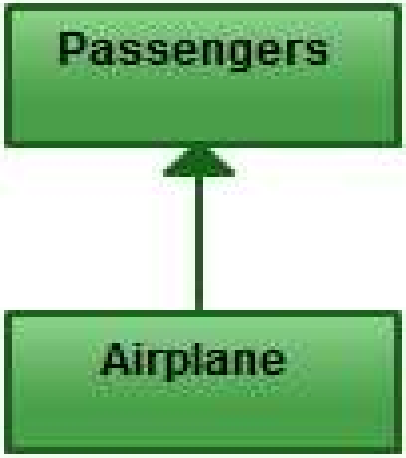

Airline Staff
0..*
0..*


Course
prerequisite
*
successor
*

Directed Association

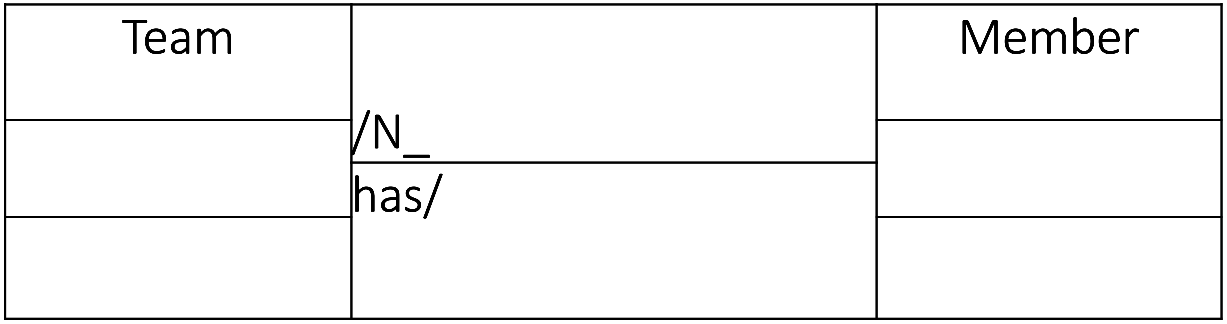
refers to a directional relationship represented by a line with an arrowhead.

The arrowhead depicts a container-contained directional flow.



Aggregation

- has-a relationship


Team
/N_
has/
Member

Aggregation

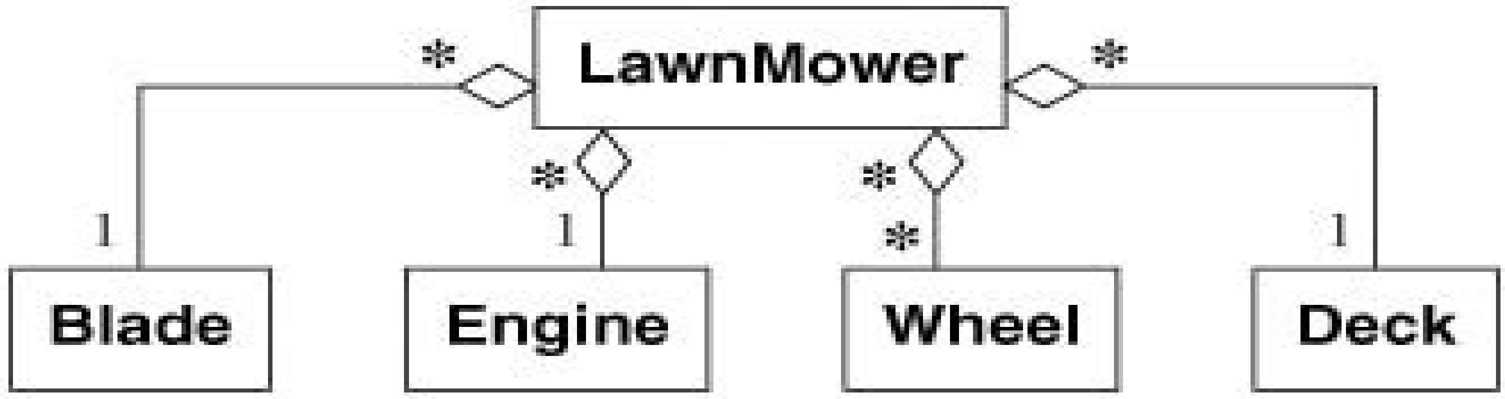
- Aggregation is a strong form of association in which an aggregate object is made of constituent parts.

- The aggregate is semantically an extended object that is treated as a UNIT in many operations, although physically it is made of several lesser objects.

- Aggregation is a transitive relation:
  - if A is a part of B and B is a part of C then A is also a part of C

- Aggregation is an antisymmetric relation:
  - If A is a part of B then B is not a part of A.

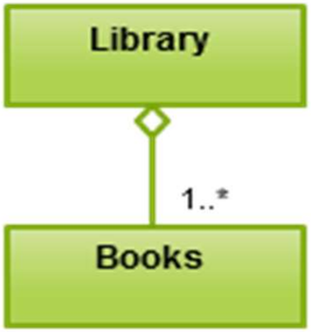
Aggregation

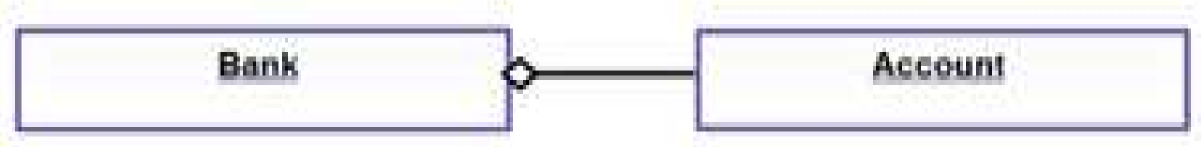

LawnMower
Blade
Engine
Wheel
Deck
*
1
*
1
*
*
*
1

Aggregation

- refers to the formation of a particular class as a result of one class being aggregated or built as a collection.
- For example, the class “library” is made up of one or more books, among other materials.

- In aggregation, the contained classes are not strongly dependent on the lifecycle of the container.
- In the same example, books will remain so even when the library is dissolved.
- To show aggregation in a diagram, draw a line from the parent class to the child class with a diamond shape near the parent class.





Aggregation versus Association

- Aggregation is a special form of association, not an independent concept.

- Aggregation adds semantic connotations:

  - If two objects are tightly bound by a part-whole relation it is an aggregation.

  - If the two objects are usually considered as independent, even though they may often be linked, it is an association.

Aggregation versus Association

- Discovering aggregation
  - Would you use the phrase part of ?
  - Do some operations on the whole automatically apply to its parts?
  - Do some attributes values propagates from the whole to all or some parts?
  - Is there an asymmetry to the association, where one class is subordinate to the other?

Aggregation versus Composition

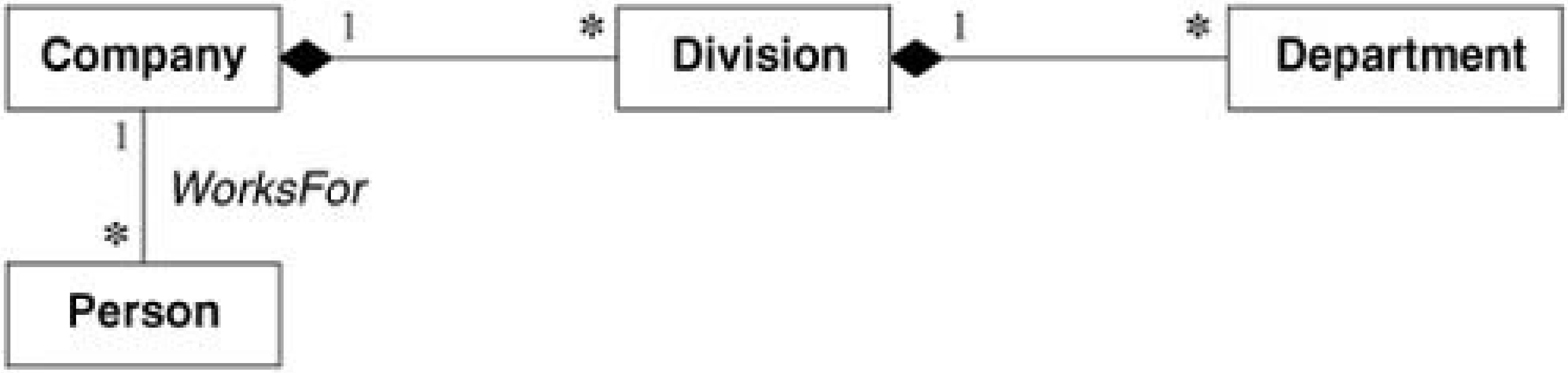
- **Composition** is a form of aggregation with additional constraints:
    - A constituent part can belong to **at most one** assembly (whole).
        - it has a coincident lifetime with the assembly.
        - Deletion of an assembly object triggers automatically a deletion of all constituent objects via composition.
    - Composition implies ownership of the parts by the whole.
        - Parts cannot be shared by different wholes.

Aggregation versus Composition

Composition is a strong form of aggregation where:

- Deletion of a “whole” triggers automatically deletion of all its “parts”. (Lifetime)
- “Parts” are included in at most one “whole” at a time. (Sharing)

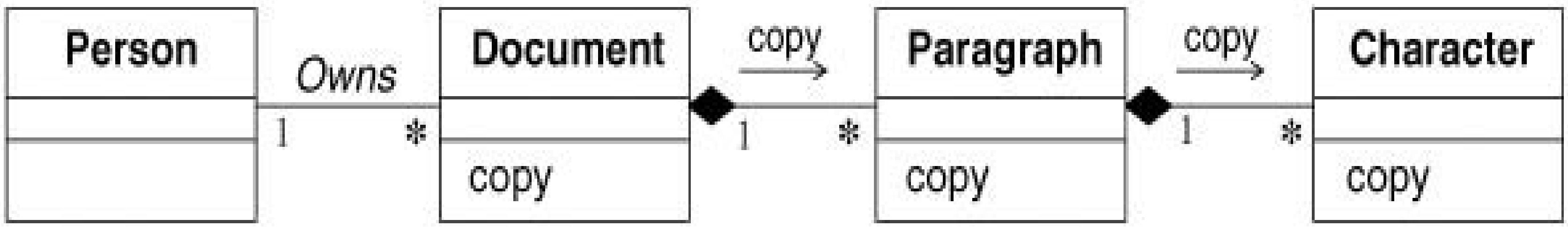
Aggregation versus Composition


Company
1
*
Division
1
*
Department
1
WorksFor
*
Person

Propagation of operations

- Propagation is the automatic application of an operation to a network of objects when the operation is applied to some starting object.

- Propagation of operations to parts is often a good indicator of propagation.

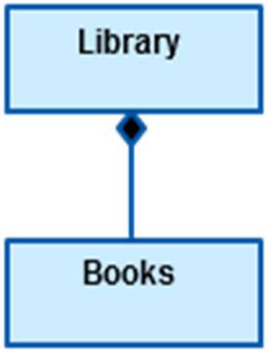

Person
Owns
1
*
Document
copy
copy
1
*
Paragraph
copy
copy
1
*
Character
copy

Composition

"is-composed-of" relationship

Composition

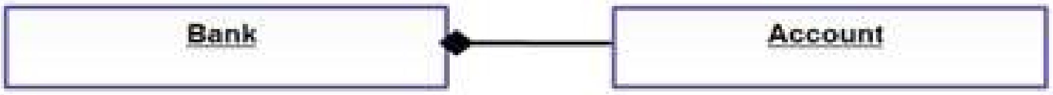
- The composition relationship is very similar to the aggregation relationship. with the only difference being its key purpose of emphasizing the dependence of the contained class to the life cycle of the container class.



- That is, the contained class will be obliterated when the container class is destroyed.

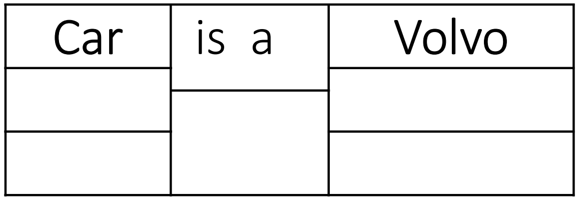
- For example, a shoulder bag’s side pocket will also cease to exist once the shoulder bag is destroyed.

- To show a composition relationship in a UML diagram, use a directional line connecting the two classes, with a filled diamond shape adjacent to the container class and the directional arrow to the contained class.



Generalization

"is-a-kind-of" relationship



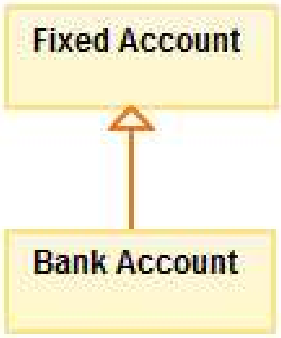
Car is a Volvo

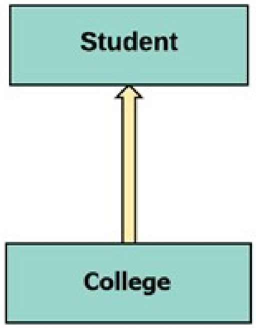
Generalization

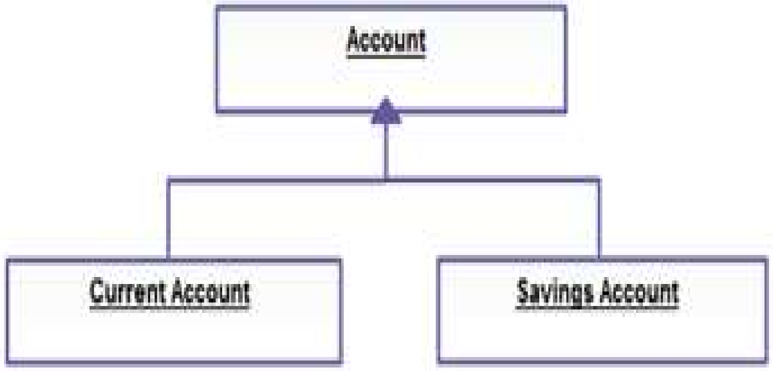
- refers to a type of relationship wherein one associated class is a child of another by virtue of assuming the same functionalities of the parent class.

- In other words, the child class is a specific type of the parent class.

- To show inheritance in a UML diagram, a solid line from the child class to the parent class is drawn using an unfilled arrowhead.


Fixed Account
Bank Account


Student
College


Account
Current Account
Savings Account

Generalization/Inheritance

- Generalization is the relationship between a class (superclass) and one or more variations of the class (subclasses).

- Generalization organizes classes by their similarities and their differences, structuring the descriptions of objects.

- A superclass holds common attributes, attributes and associations.

- The subclasses adds specific attributes, operations, and associations. They inherit the features of their superclass.

Generalization/Inheritance

- Often **Generalization** is called a “**IS A**” relationship

- **Simple generalization** organizes classes into a **hierarchy**.

- A subclass may **override** a superclass **feature** (attribute default values, operation) by **redefining a feature with the same name.**

- Never override the signature of methods.

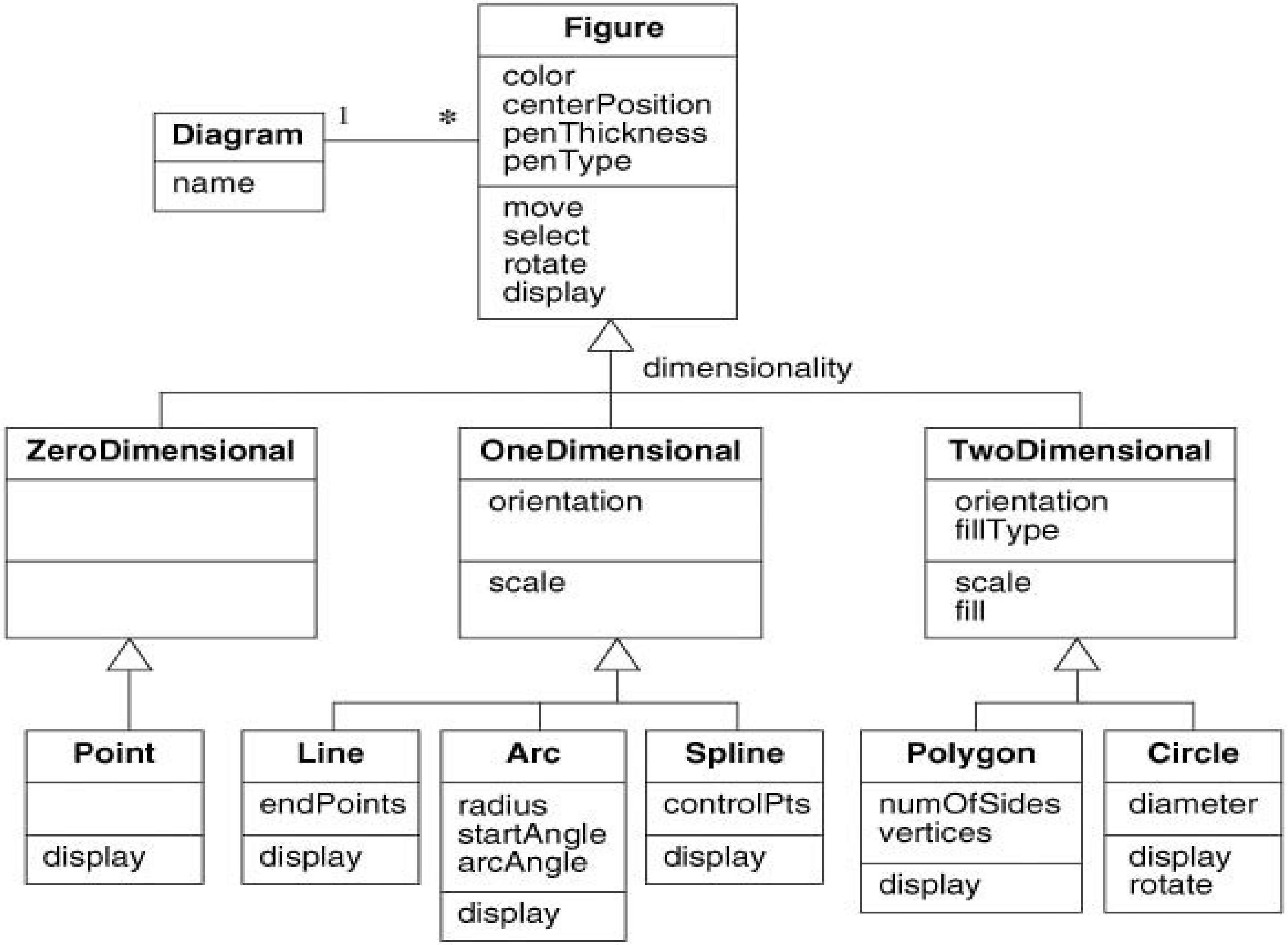


Figure
color
centerPosition
penThickness
penType
move
select
rotate
display
Diagram
name
1
*
dimensionality
ZeroDimensional
OneDimensional
orientation
scale
TwoDimensional
orientation
fillType
scale
fill
Point
display
Line
endPoints
display
Arc
radius
startAngle
arcAngle
display
Spline
controlPts
display
Polygon
numOfSides
vertices
display
Circle
diameter
display
rotate

Use of generalization

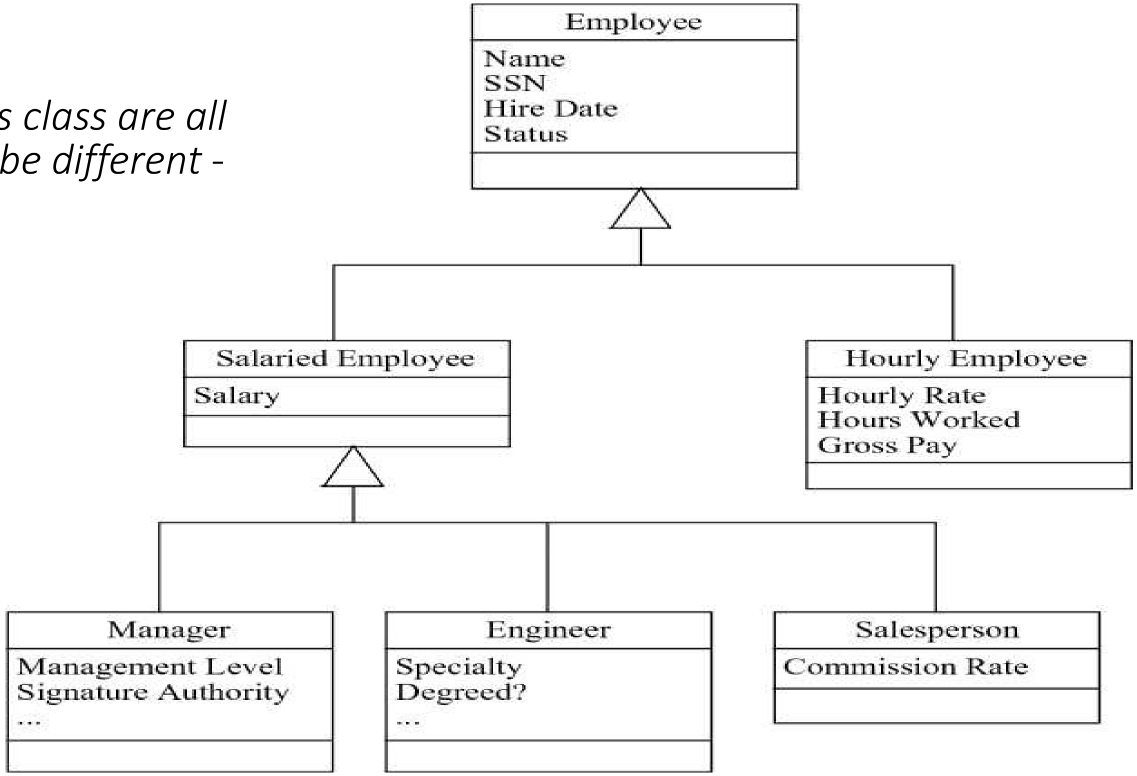
- Used for three purposes:
  - Support of polymorphism:
    - polymorphism increases the flexibility of software.
    - Adding a new subclass and automatically inheriting superclass behavior.
  - Structuring the description of objects:
    - Forming a taxonomy (classification), organizing objects according to their similarities.
    - It is much more profound than modeling each class individually and in isolation of other similar classes.
  - Enabling code reuse:
    - Reuse is more productive than repeatedly writing code from scratch.

Generalization

- Generalization is a way of saying

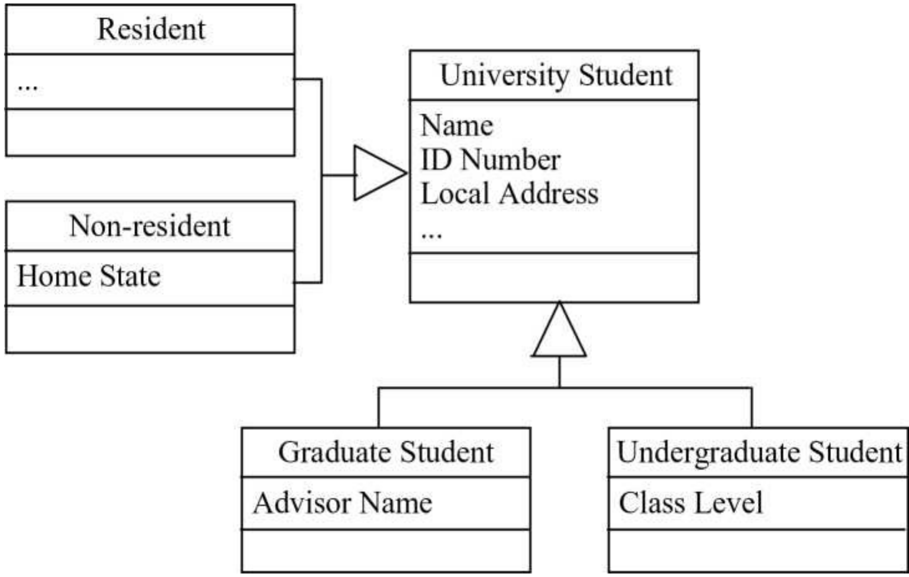
  At one level of abstraction, the objects in this class are all the same but at a finer level of detail, they may be different - Represents an is-a or is-a-kind-of relationship

- Notation


Employee
Name
SSN
Hire Date
Status
Salaried Employee
Salary
Hourly Employee
Hourly Rate
Hours Worked
Gross Pay
Manager
Management Level
Signature Authority
...
Engineer
Specialty
Degreed?
...
Salesperson
Commission Rate

Generalization (cont..)

- The subclasses may represent a complete or incomplete breakdown of the superset (are all subsets represented?)
  - Use the constraint {complete} or {incomplete}
- The subsets may represent (non-)mutually exclusive groupings
  - Use the constraint {disjoint} or {overlapping}


Resident
...
Non-resident
Home State
University Student
Name
ID Number
Local Address
...
Graduate Student
Advisor Name
Undergraduate Student
Class Level

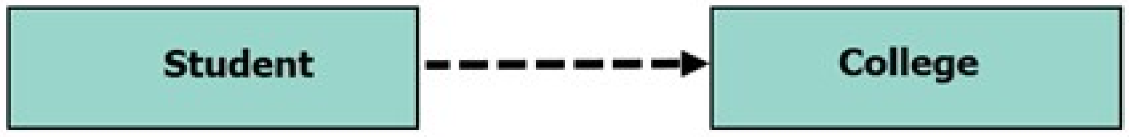
- Multiple and Dynamic Classification
  - Multiple classification means that the same superset can be specialized in multiple dimensions
  - Dynamic classification means that an instance of a superset can change from one subset to another

Dependency

A dependency means the relation between two or more classes in which a change in one may force changes in the other.

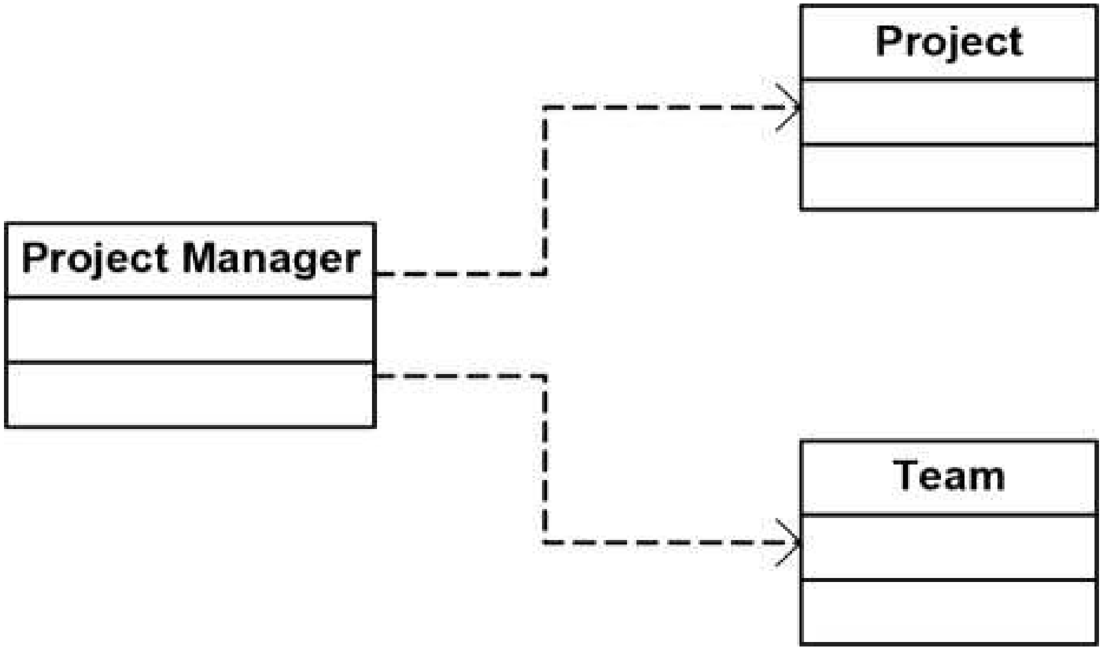
However, it will always create a weaker relationship. Dependency indicates that one class depends on another.

In the following UML class diagram examples, Student has a dependency on College



Dependency

The source class depends on (uses) the target class, (not used for requirements analysis)

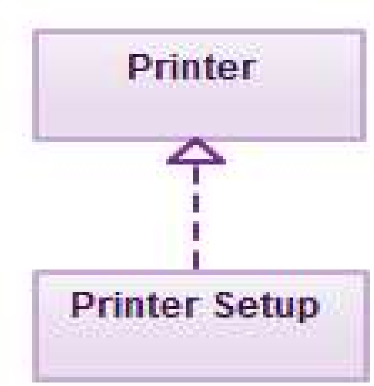

Project
Project Manager
Team

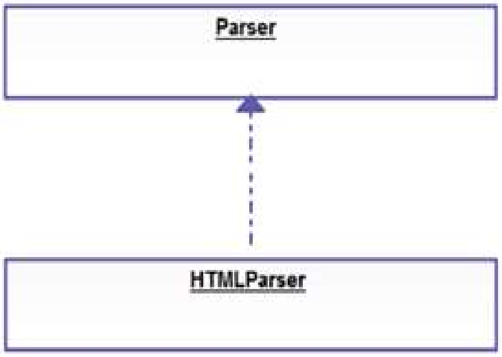
Realization

denotes the implementation of the functionality defined in one class by another class.

To show the relationship in UML, a broken line with an unfilled solid arrowhead is drawn from the class that defines the functionality of the class that implements the function.

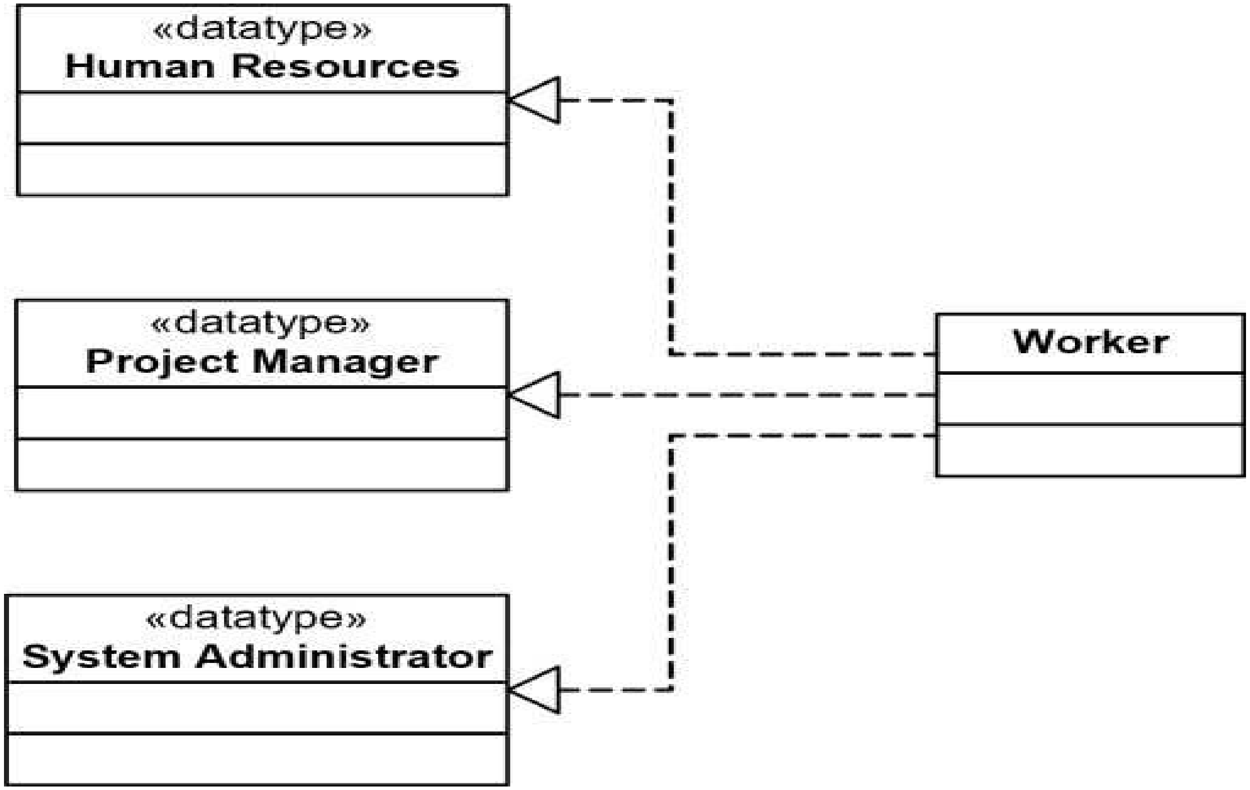
In the example, the printing preferences that are set using the printer setup interface are being implemented by the printer.



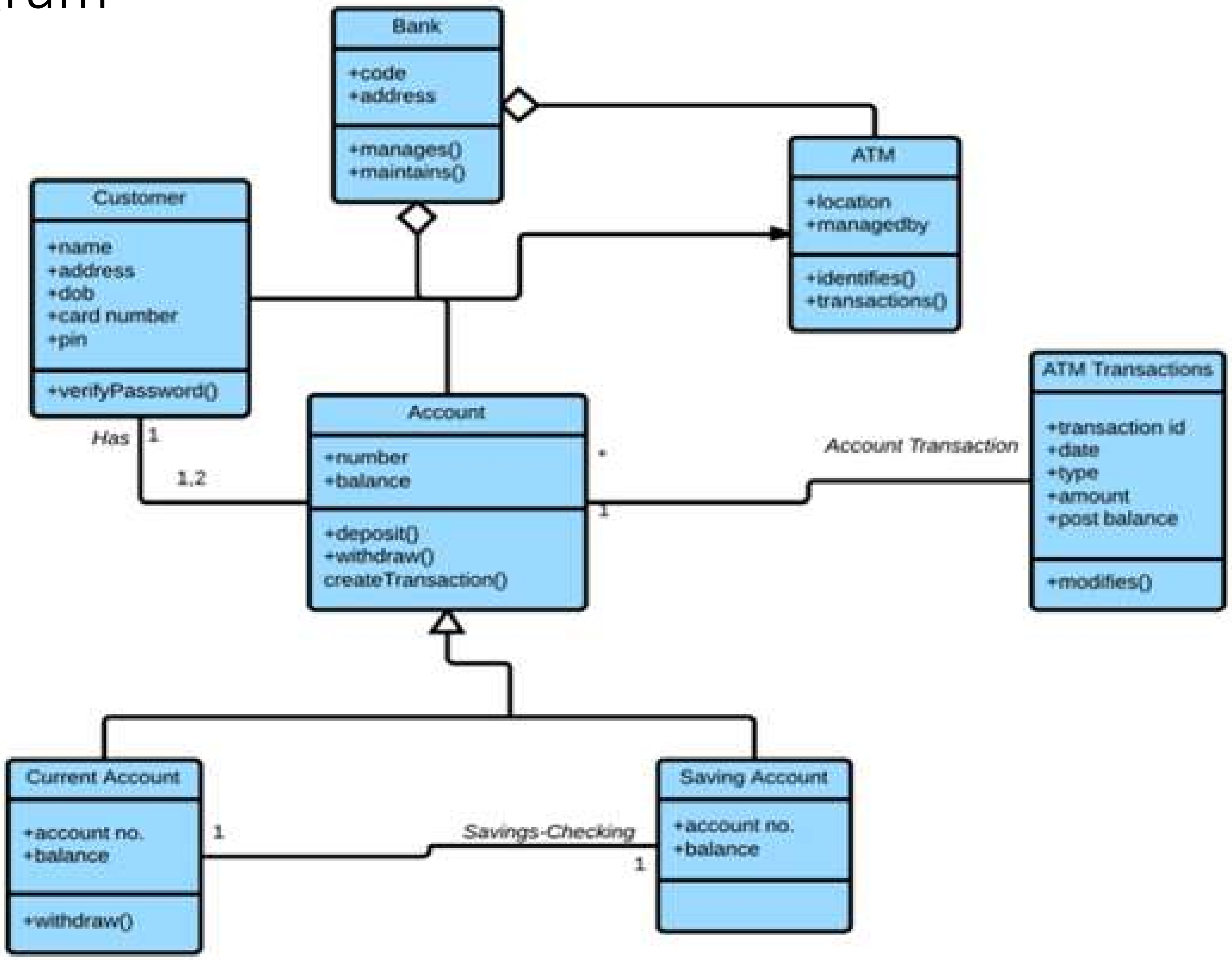


Realization

Class supports all operations of target class but not all attributes or associations, (not used for requirements analysis)


«datatype»
Human Resources
«datatype»
Project Manager
«datatype»
System Administrator
Worker

Example of Class Diagram


Bank
+code
+address
+manages()
+maintains()
ATM
+location
+managedby
+identifies()
+transactions()
Customer
+name
+address
+dob
+card number
+pin
+verifyPassword()
Has
1
1,2
Account
+number
+balance
+deposit()
+withdraw()
createTransaction()
*
1
Account Transaction
ATM Transactions
+transaction id
+date
+type
+amount
+post balance
+modifies()
Current Account
+account no.
+balance
+withdraw()
1
Savings-Checking
1
Saving Account
+account no.
+balance

Class Diagram for Library Management System :

Aggregation and Multiplicity are two important points that need to take into consideration while designing a Class Diagram.

Let us understand in detail.

1. Aggregation –
   Aggregation simply shows a relationship where one thing can exist independently of other thing.

It means to create or compose different abstractions together in defining a class.

Aggregation is represented as a part of relationship in class diagram.

Class Diagram for Library Management System :

In diagram given below, we can see that aggregation is represented by an edge with a diamond end pointing towards superclass.

The “Library Management System” is superclass that consists of various classes.

These classes are User, Book, and Librarian as shown in diagram.

Further, for “Account” class, “User” is a superclass.

All of these, share a relationship and these relationships are known as aggregate relationships.

2. Multiplicity –

Multiplicity means that number of elements of a class is associated with another class.

- These relations can be one-to-one, many-to-many, and many-to-one or one-to-many.
- For denoting one element we use 1, for zero elements we use 0, and for many elements we use *.
- We can see in diagram; many users are associated with many books denoted by * and this represents a many-to-many type of relationship.

2. Multiplicity –

One user has only one account that is denoted by 1 and this represents a one-to-one type of relationship.

Many books are associated with one librarian and this represents many-to-one or one-to-many type of relationship.

All these relationships are shown in diagram.

Class Diagram for Library Management System simply describes structure of Library Management System class, attributes, methods or operations, relationship among objects.

Classes for Library Management System :

- Library Management System class –
  It manages all operations of Library Management System.
  It is central part of organization for which software is being designed.

- User Class –
  It manages all operations of user.

- Librarian Class – It manages all operations of Librarian.

- Book Class –
  It manages all operations of books. It is basic building block of system.

Classes for Library Management System :

- Account Class –
  It manages all operations of account.

- Library database Class –
  It manages all operations of library database.

- Staff Class –
  It manages all operations of staff.

- Student Class –
  It manages all operations of student.

Attributes for Library Management System :

- Library Management System Attributes –
  UserType, Username, Password

- User Attributes –
  Name, Id

- Librarian Attributes –
  Name, Id, Password, SearchString

- Book Attributes –
  Title, Author, ISBN, Publication

Attributes for Library Management System :

- Account Attributes –
  no_borrowed_books, no_reserved_books, no_returned_books, no_lost_books fine_amount

- Library database Attributes –
  List_of_books

- Staff Class Attributes –
  Dept

- Student Class Attributes –
  Class

Methods of Library Management System :

- Library Management System Methods –
  Login(), Register(), Logout()

- User Methods –
  Verify(), CheckAccount(), get_book_info()

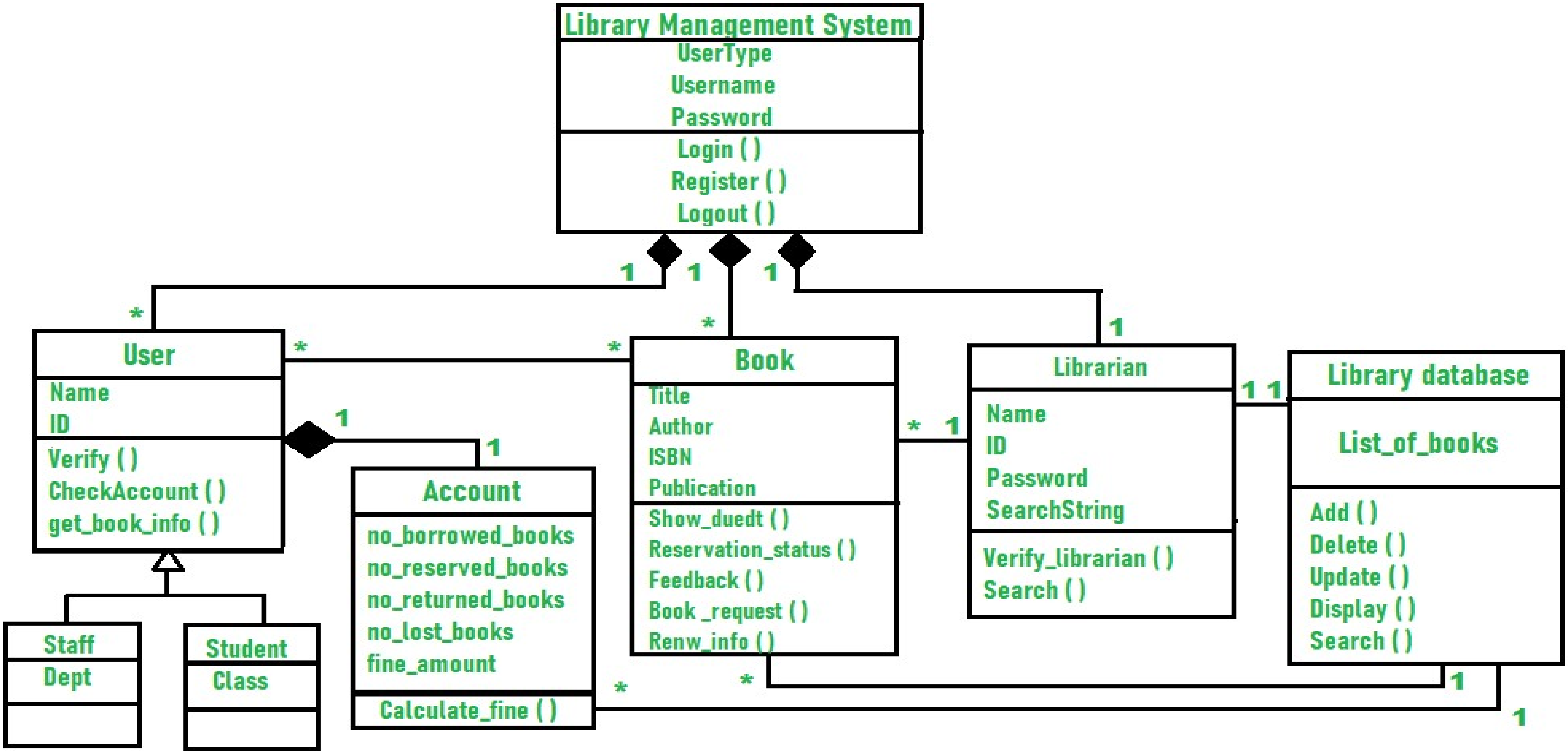
- Librarian Methods –
  Verify_librarian(), Search()

Methods of Library Management System :

- Book Methods –
  Show_duedt(), Reservation_status(), Feedback(), Book_request(), Renew_info()

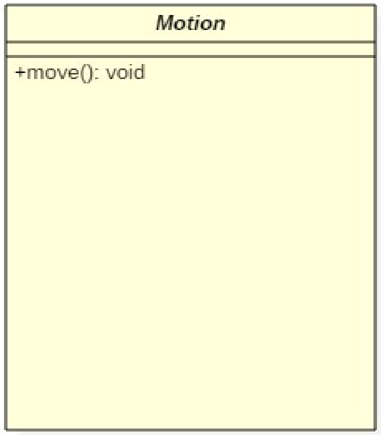
- Account Methods –
  Calculate_fine()

- Library database Methods –
  Add(), Delete(), Update(), Display(), Search()


Library Management System
UserType
Username
Password
Login ( )
Register ( )
Logout ( )
1
1
1
*
*
*
1
User
Name
ID
Verify ( )
CheckAccount ( )
get_book_info ( )
*
1
1
Book
Title
Author
ISBN
Publication
Show_duedt ( )
Reservation_status ( )
Feedback ( )
Book _request ( )
Renw_info ( )
*
1
Librarian
Name
ID
Password
SearchString
Verify_librarian ( )
Search ( )
1 1
Library database
List_of_books
Add ( )
Delete ( )
Update ( )
Display ( )
Search ( )
Account
no_borrowed_books
no_reserved_books
no_returned_books
no_lost_books
fine_amount
Calculate_fine ( )
Staff
Dept
Student
Class
*
*
1
1

CLASS DIAGRAM FOR LIBRARY MANAGEMENT SYSTEM

An abstract class



Abstract Class Notation

It is a class with an operation prototype, but not the implementation.

It is also possible to have an abstract class with no operations declared inside of it.

An abstract is useful for identifying the functionalities across the classes.

Let us consider an example of an abstract class. Suppose we have an abstract class called as a motion with a method or an operation declared inside of it.

The method declared inside the abstract class is called a move ().

This abstract class method can be used by any object such as a car, an animal, robot, etc. for changing the current position.

An abstract class

- It is efficient to use this abstract class method with an object because no implementation is provided for the given function.

- We can use it in any way for multiple objects.

- In UML, the abstract class has the same notation as that of the class.

- The only difference between a class and an abstract class is that the class name is strictly written in an italic font.

- An abstract class cannot be initialized or instantiated.

Key Points

- The class diagram is the main “internal” view

- The focus is on static structure, things about the business policy/business process that should (effectively) always be true

- A Class is the descriptor for a set of objects with similar structure, behavior, and relationships

- An Attribute is a single, named fragment of the persistent state

- An Association exists when some number of objects are connected (linked) in some significant manner and it is necessary to remember that connection

Key Points

- An Association Role indicates the role (purpose) that the class plays in the association

- Multiplicity describes the rules about the number of objects in each class that may or may not be linked

- Aggregation is a kind of association that represents a whole-part relationship

- Composition is like aggregation with the added constraint that the part may be the part of exactly one whole

- Associations may take on the characteristics of a class

- Generalization is a way of saying, “At one level of abstraction, the objects in this class are all the same but at a finer level of detail, they may be different”

Key Points

some points which should be kept in mind while drawing a class diagram:

- The name given to the class diagram must be meaningful.
- Moreover, It should describe the real aspect of the system.
- The relationship between each element needs to be identified in advance.
- The responsibility for every class needs to be identified.

Key Points

- For every class, minimum number of properties should be specified.
- Therefore, unwanted properties can easily make the diagram complicated.
- User notes should be included whenever you need to define some aspect of the diagram.
- At the end of the drawing, it must be understandable for the software development team.
- Lastly, before creating the final version, the diagram needs to be drawn on plain paper.
- Moreover, It should be reworked until it is ready for final submission.

Conclusion

- UML is the standard language for specifying, designing, and visualizing the artifacts of software systems

- A class is a blueprint for an object

- A class diagram describes the types of objects in the system and the different kinds of relationships which exist among them

- It allows analysis and design of the static view of a software application

- Class diagrams are most important UML diagrams used for software application development

Conclusion

- Essential elements of UML class diagram are :
- 1) Class
- 2) Attributes
- 3) Relationships

- Class Diagram provides an overview of how the application is structured before studying the actual code.
- It certainly reduces the maintenance time
- The class diagram is useful to map object-oriented programming languages like Java, C++, Ruby, Python, etc.

BITS Pilani
Pilani Campus

SOFTWARE DESIGN PRINCILES

Unified Modeling Language for Dynamic Analysis

Prof A R Rahman

Dynamic Behaviour

- This interaction is a part of dynamic behavior of the system.
- This interactive behavior is represented in UML by two diagrams known as Sequence diagram and Collaboration diagram.
- Sequence diagram emphasizes on time sequence of messages
- Collaboration diagram emphasizes on the structural organization of the objects that send and receive messages.

Dynamic Behaviour

Interaction Diagrams - Purpose

- The purpose of interaction diagrams is to visualize the interactive behavior of the system.

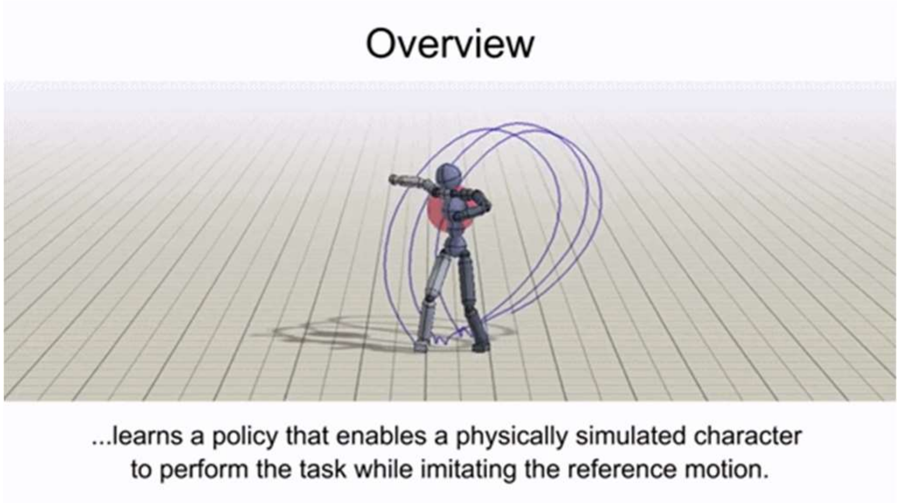
- Visualizing the interaction is a difficult task.

- Hence, the solution is to use different types of models to capture the different aspects of the interaction.

- Sequence and collaboration diagrams are used to capture the dynamic nature but from a different angle.

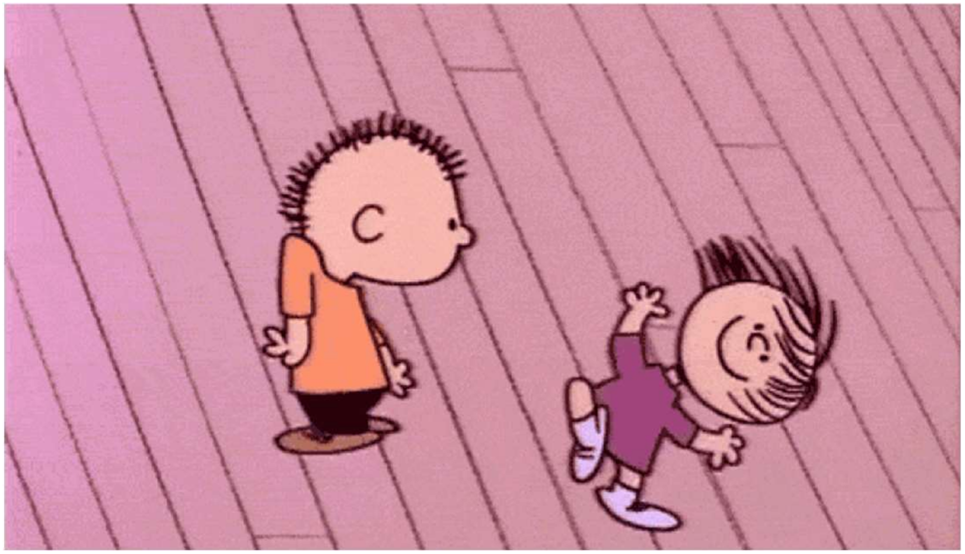
The purpose of interaction diagram is –

- To capture the dynamic behaviour of a system.
- To describe the message flow in the system.
- To describe the structural organization of the objects.
- To describe the interaction among objects.


Overview
...learns a policy that enables a physically simulated character to perform the task while imitating the reference motion.

How to Draw an Interaction Diagram?

- The purpose of interaction diagrams is to capture the dynamic aspect of a system.

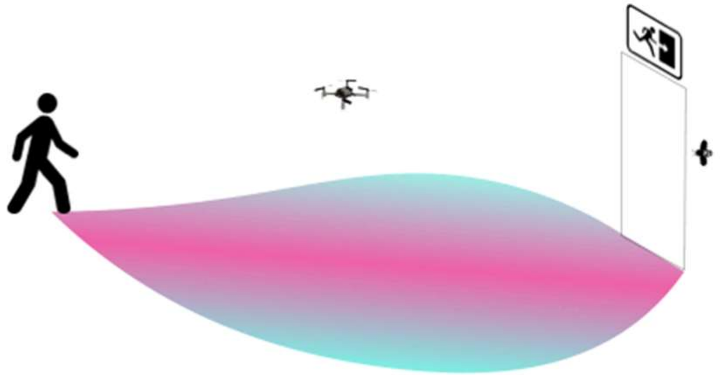


- So to capture the dynamic aspect, we need to understand what a dynamic aspect is and how it is visualized.

- Dynamic aspect can be defined as the snapshot of the running system at a particular moment

Interaction Diagram

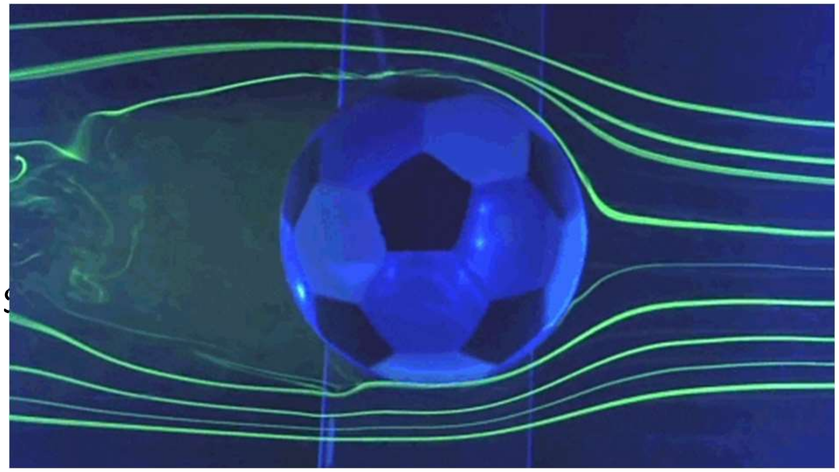
- We have two types of interaction diagrams in UML.
- One is the sequence diagram and the other is the collaboration diagram.
- The sequence diagram captures the time sequence of the message flow from one object to another and the collaboration diagram describes the organization of objects in a system taking part in the message flow.



Interaction Diagram

- Following things are to be identified clearly before drawing the interaction diagram
- Objects taking part in the interaction.
- Message flows among the objects.
- The sequence in which the messages are flowing.
- Object organization.

Interaction Diagrams



- show how objects interact with one another

- UML supports two types of interaction diagrams
  - Sequence diagrams
  - Collaboration diagrams

- There are 3 types of Interaction diagrams; Sequence diagrams, communication diagrams, and timing diagrams.

- These diagrams are used to illustrate interactions between parts within a system.

- Among the three, sequence diagrams are preferred by both developers and readers alike for their simplicity.

Interaction Diagrams

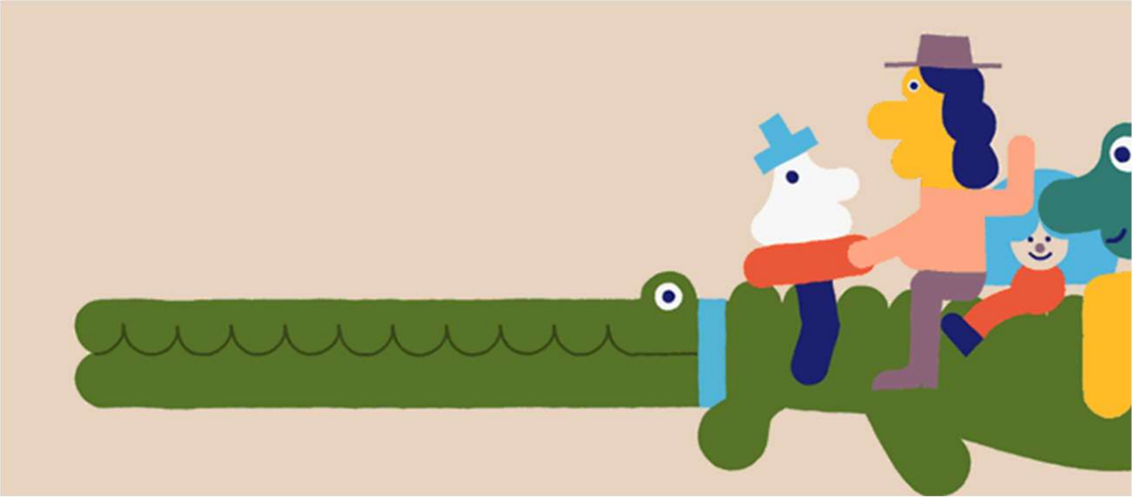
- UML Specifies a number of interaction diagrams to model dynamic aspects of the system

- Dynamic aspects of the system

  - Messages moving among objects/ classes

  - Flow of control among objects

  - Sequences of events



Dynamic Diagram Types

- Interaction Diagrams - Set of objects or roles and the messages that can be passed among them.

  - Sequence Diagrams - emphasize time ordering
  - Communication Diagrams - emphasize structural orde

- State Diagrams
  - State machine consisting of states, transitions, events activities of an object

- Activity & Swimlane Diagrams
  - Emphasize and show flow of control among objects

Error
Wifi no connected dumb dumb

What is a Sequence Diagram?

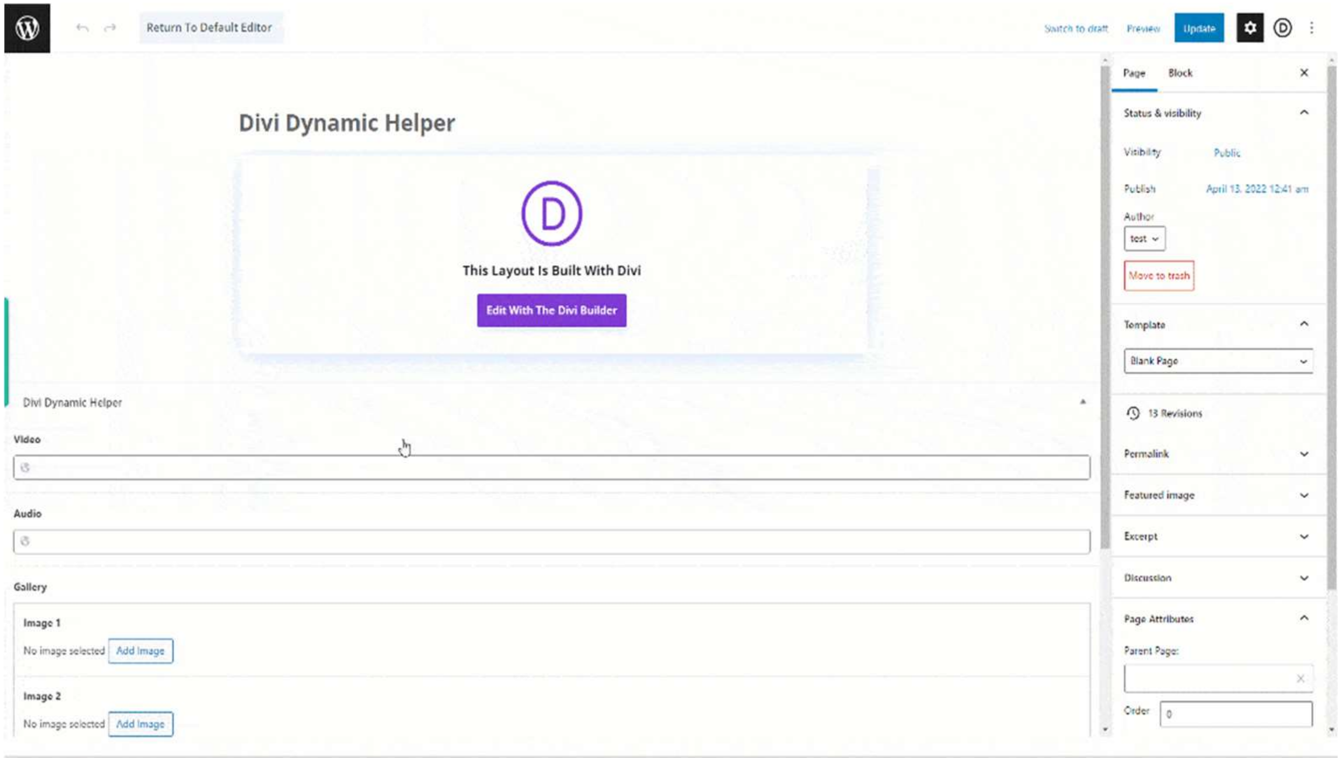
- Sequence diagrams, commonly used by developers, model the interactions between objects in a single use case.

- They illustrate how the different parts of a system interact with each other to carry out a function, and the order in which the interactions occur when a particular use case is executed.

- In simpler words, a sequence diagram shows different parts of a system work in a 'sequence' to get something done.

Sequence Diagram Notations

- A sequence diagram is structured in such a way that it represents a timeline which begins at the top and descends gradually to mark the sequence of interactions.
- Each object has a column and the messages exchanged between them are represented by arrows.


Return To Default Editor
Switch to draft
Preview
Update
Page
Block
Divi Dynamic Helper
This Layout Is Built With Divi
Edit With The Divi Builder
Divi Dynamic Helper
Video
Audio
Gallery
Image 1
No image selected
Add Image
Image 2
No image selected
Add Image
Status & visibility
Move to trash
Template
13 Revisions
Permalink
Featured image
Excerpt
Discussion
Page Attributes
Parent Page:

Visibility Public
Publish April 13, 2022 12:41 am
Author test
Template Blank Page
Order 0

Sequence Diagrams

- Describe the flow of messages, events, actions between objects
- Show concurrent processes and activations
- Show time sequences that are not easily depicted in other diagrams
- Typically used during analysis and design to document and understand the logical flow of your system

Emphasis on time ordering!

Where to Use Interaction Diagrams?

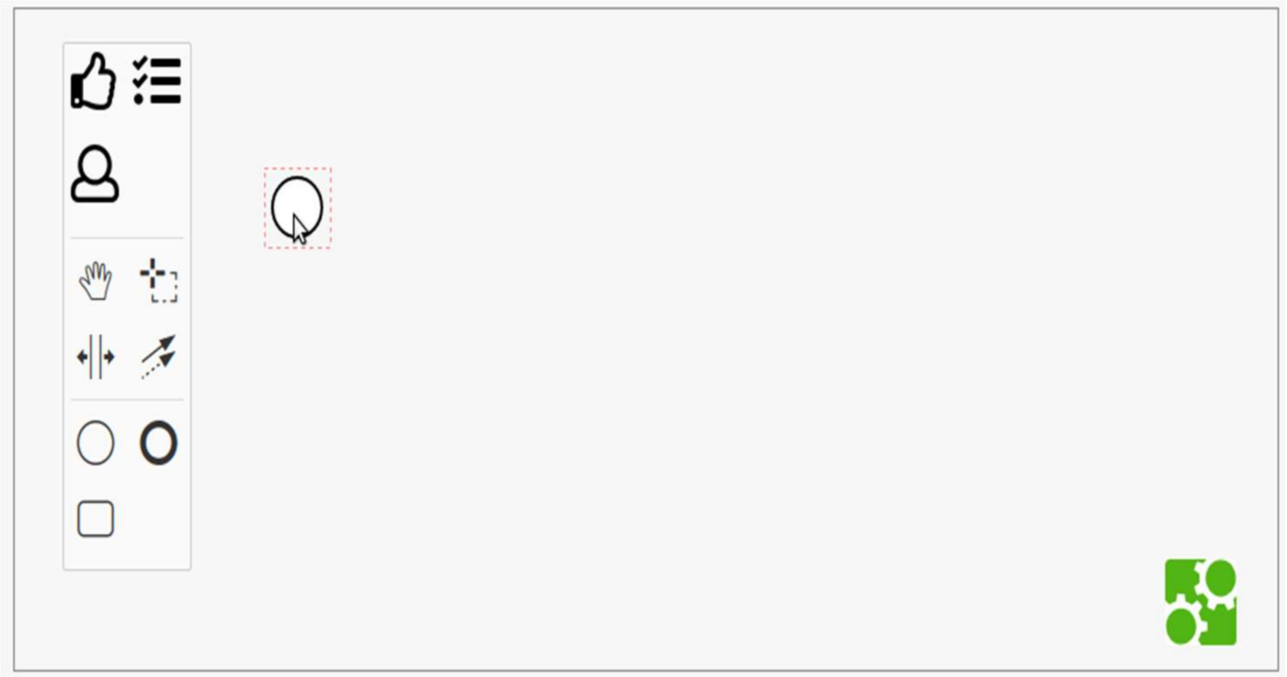
- We have already discussed that interaction diagrams are used to describe the dynamic nature of a system.

- Now, we will look into the practical scenarios where these diagrams are used.

- To understand the practical application, we need to understand the basic nature of sequence and collaboration diagram.

- The main purpose of both the diagrams are similar as they are used to capture the dynamic behavior of a system.

- Sequence diagrams are used to capture the order of messages flowing from one object to another.
- Collaboration diagrams are used to describe the structural organization of the objects taking part in the interaction.
- A single diagram is not sufficient to describe the dynamic aspect of an entire system, so a set of diagrams are used to capture it as a whole.



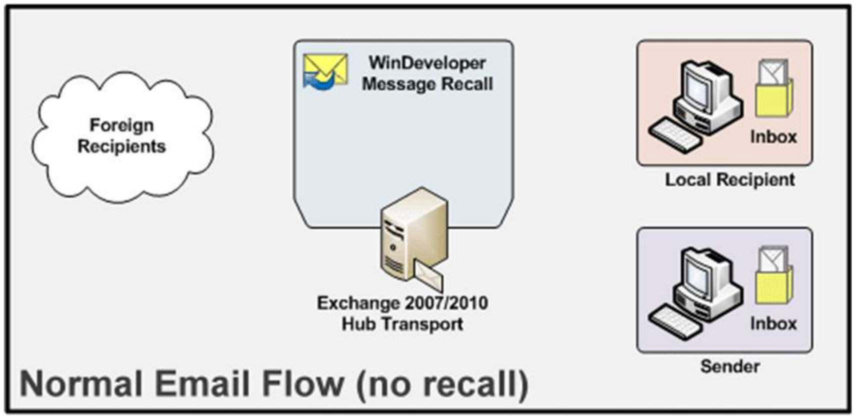
Message flow

- Message flow means the sequence of control flow from one object to another.

- Structural organization means the visual organization of the elements in a system.

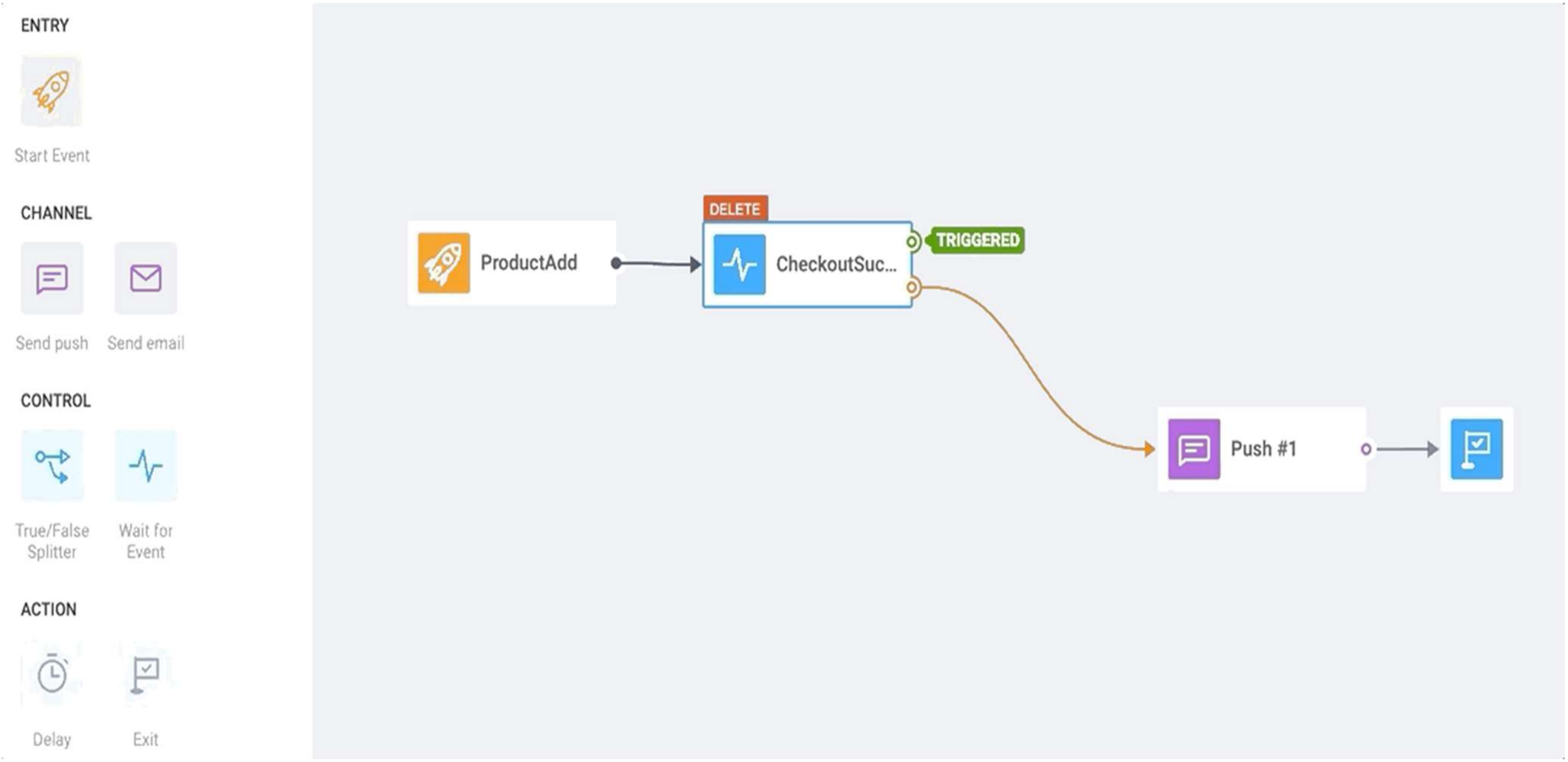
- Interaction diagrams can be used –

- To model the flow of control by time sequence.


Foreign Recipients
WinDeveloper Message Recall
Exchange 2007/2010 Hub Transport
Inbox
Local Recipient
Inbox
Sender
Normal Email Flow (no recall)

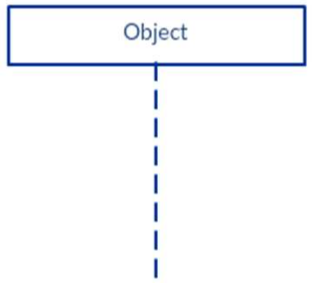
- To model the flow of control by structural organizations.
- For forward engineering.
- For reverse engineering.

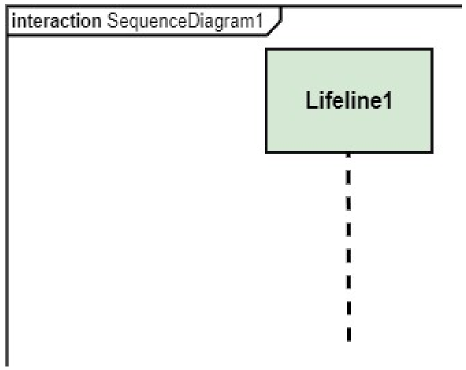
Message flow

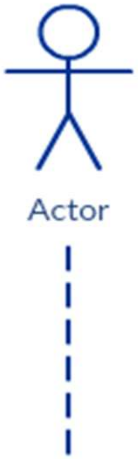

ENTRY
Start Event
CHANNEL
Send push
Send email
CONTROL
True/False Splitter
Wait for Event
ACTION
Delay
Exit
ProductAdd
DELETE
CheckoutSuc...
TRIGGERED
Push #1

Lifeline Notation

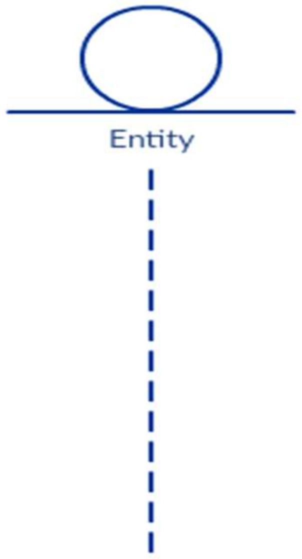
- A sequence diagram is made up of several of these lifeline notations that should be arranged horizontally across the top of the diagram.
- No two lifeline notations should overlap each other.
- They represent the different objects or parts that interact with each other in the system during the sequence.




interaction SequenceDiagram1
Lifeline1

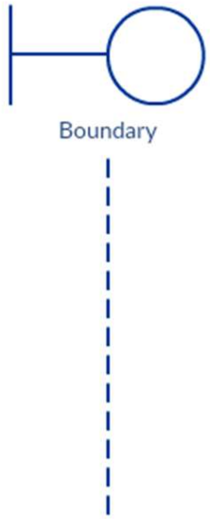


- A lifeline notation with an actor element symbol is used when the particular sequence diagram is owned by a use case.

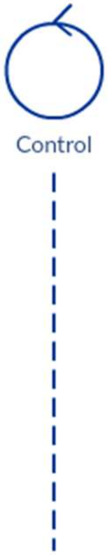


- A lifeline with an entity element represents system data.

- For example, in a customer service application, the Customer entity would manage all data related to a customer.



A lifeline with a boundary element indicates a system boundary/ software element in a system; for example, user interface screens, database gateways or menus that users interact with, are boundaries.

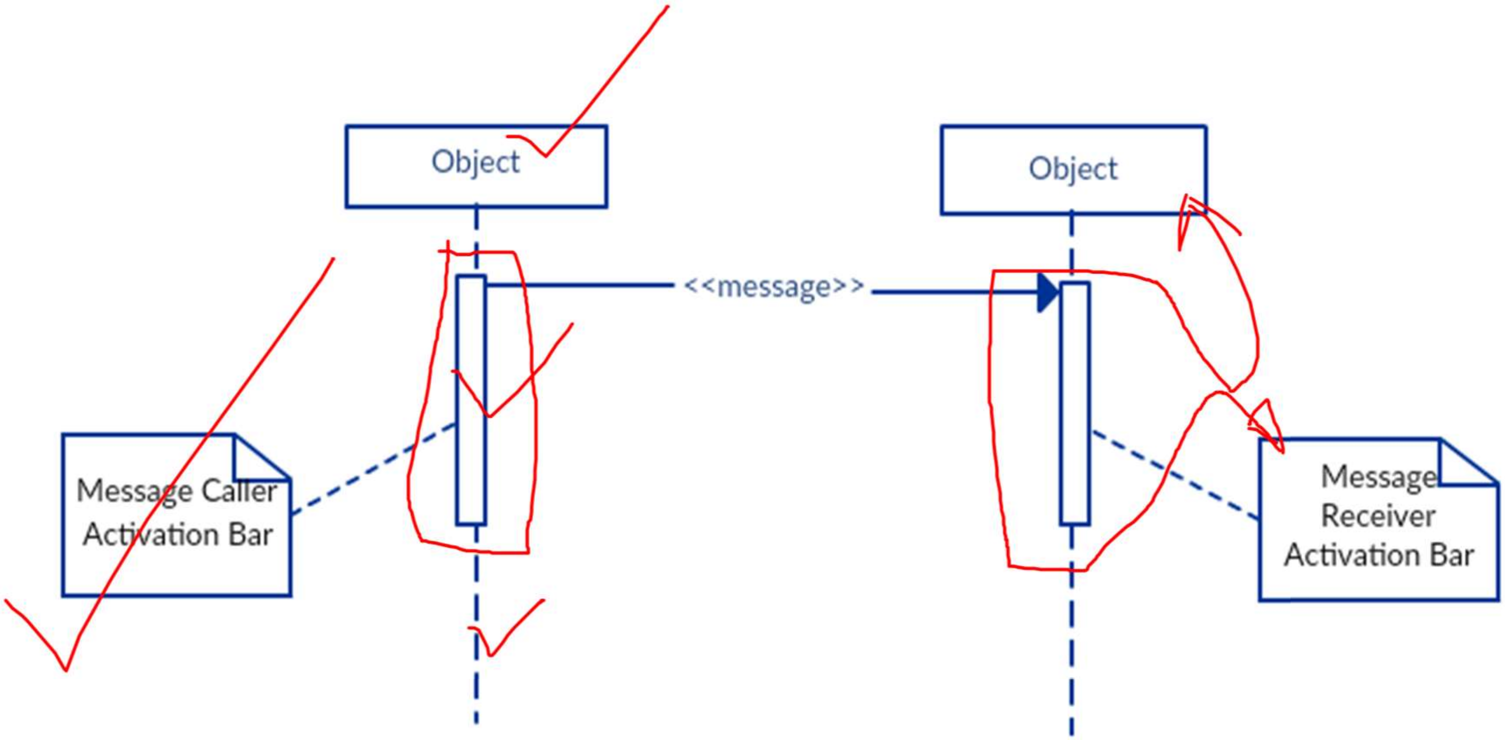


And a lifeline with a control element indicates a controlling entity or manager.

It organizes and schedules the interactions between the boundaries and entities and serves as the mediator between them.

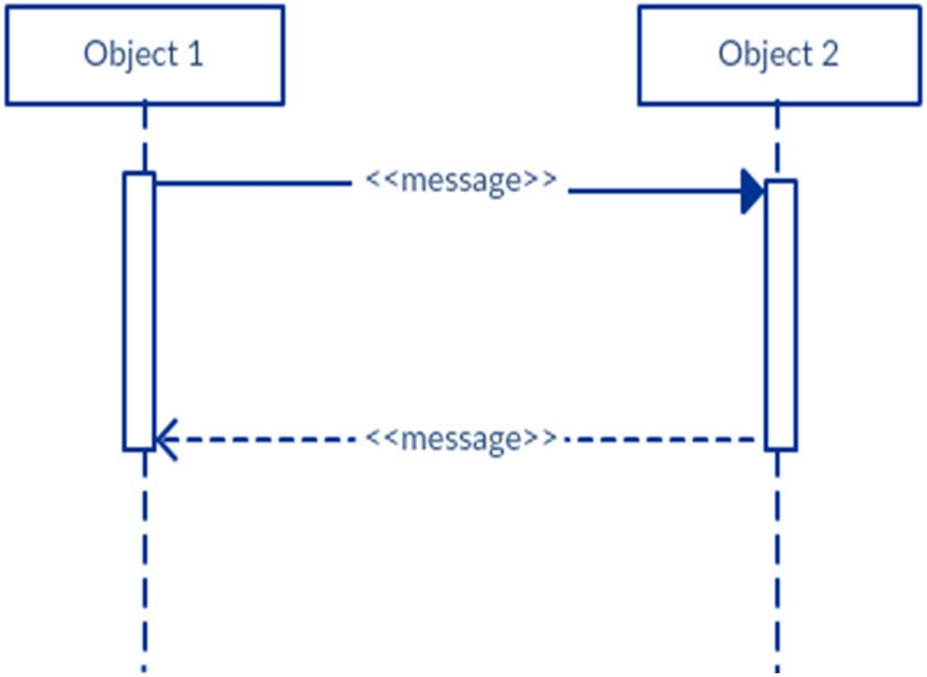
Activation Bars

- Activation bar is the box placed on the lifeline.
- It is used to indicate that an object is active (or instantiated) during an interaction between two objects.
- The length of the rectangle indicates the duration of the objects staying active.
- In a sequence diagram, an interaction between two objects occurs when one object sends a message to another.
- The use of the activation bar on the lifelines of the Message Caller (the object that sends the message) and the Message Receiver (the object that receives the message) indicates that both are active/is instantiated during the exchange of the message.


Object
Object
<<message>>
Message Caller Activation Bar
Message Receiver Activation Bar

Return message

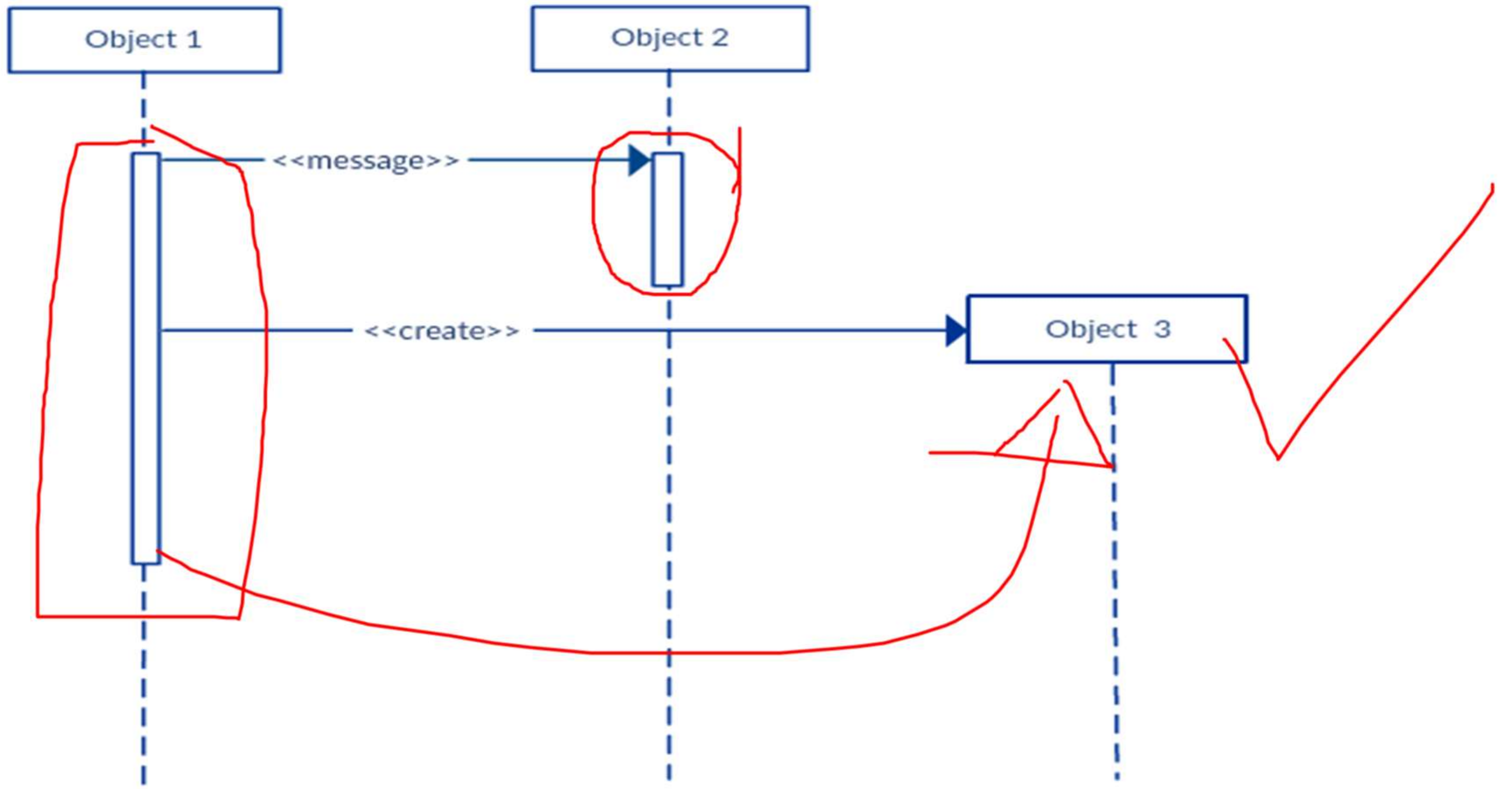
- A return message is used to indicate that the message receiver is done processing the message and is returning control over to the message caller.
- Return messages are optional notation pieces, for an activation bar that is triggered by a synchronous message always implies a return message.



Tip: You can avoid cluttering up your diagrams by minimizing the use of return messages since the return value can be specified in the initial message arrow itself.

Participant creation message

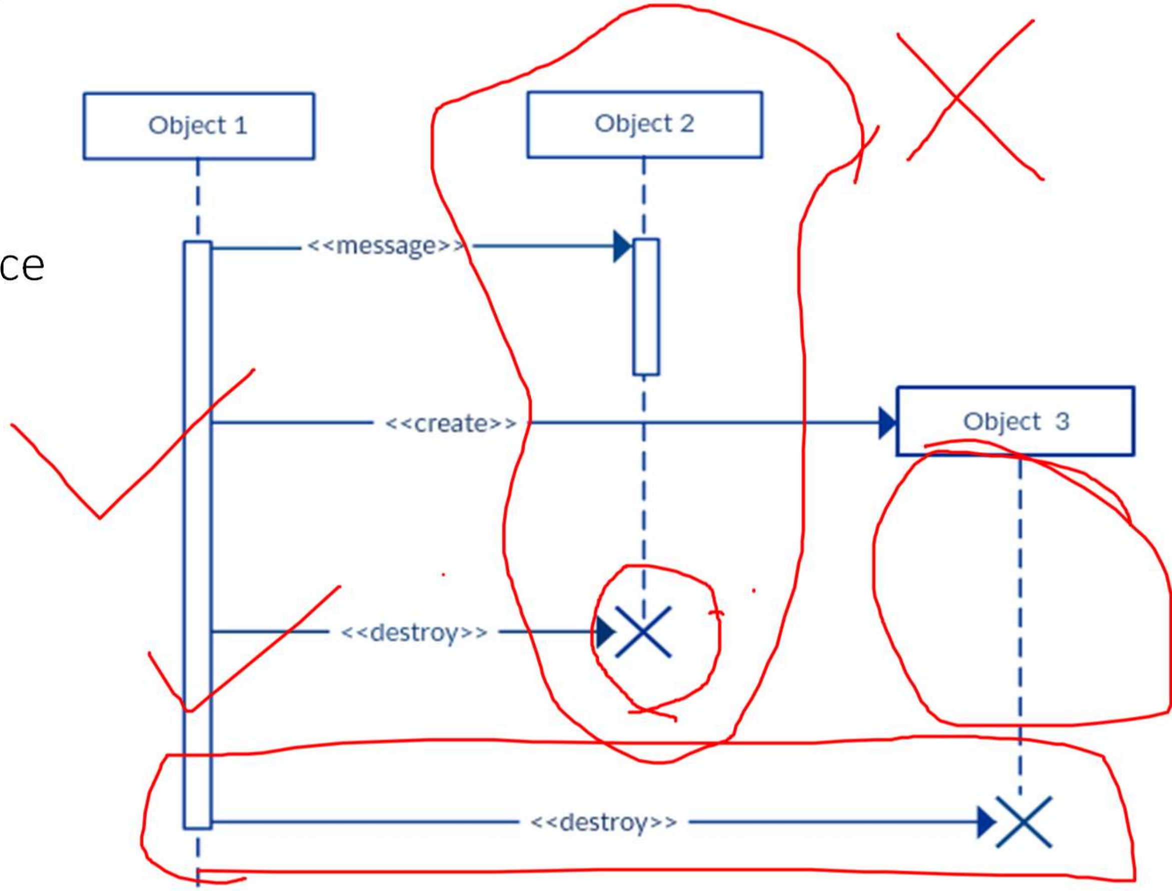
- Objects do not necessarily live for the entire duration of the sequence of events.
- Objects or participants can be created according to the message that is being sent.
- The dropped participant box notation can be used when you need to show that the particular participant did not exist until the create call was sent.
- If the created participant does something immediately after its creation, you should add an activation box right below the participant box.


Object 1
Object 2
<<message>>
<<create>>
Object 3

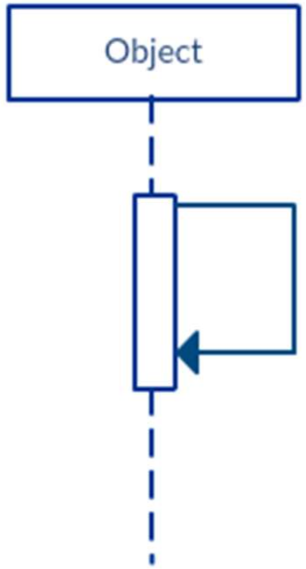
Participant destruction message

Likewise, participants when no longer needed can also be deleted from a sequence diagram.

This is done by adding an 'X' at the end of the lifeline of the said participant.



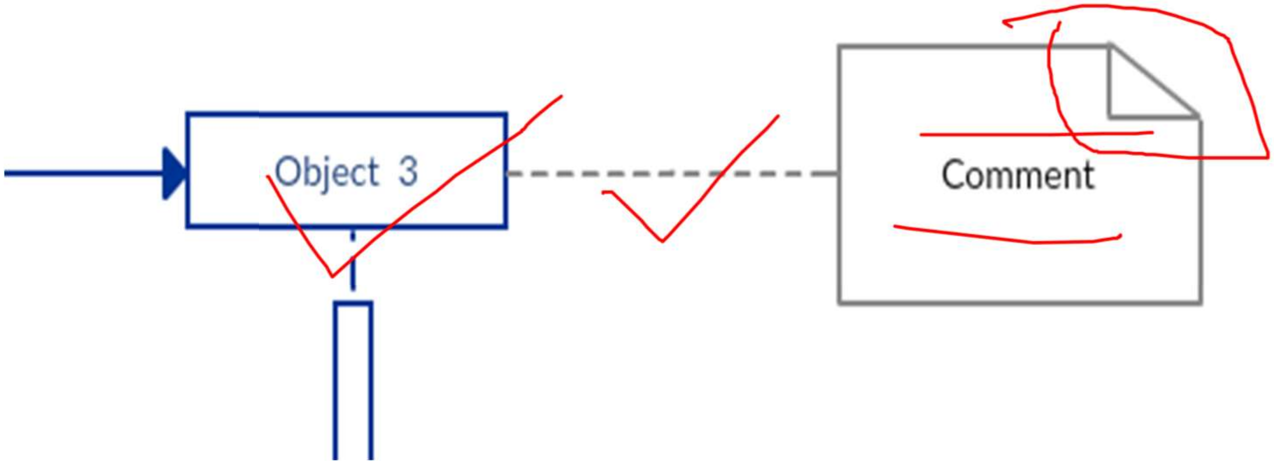
Reflexive message



When an object sends a message to itself, it is called a reflexive message.

It is indicated with a message arrow that starts and ends at the same lifeline as shown in here.

Comment

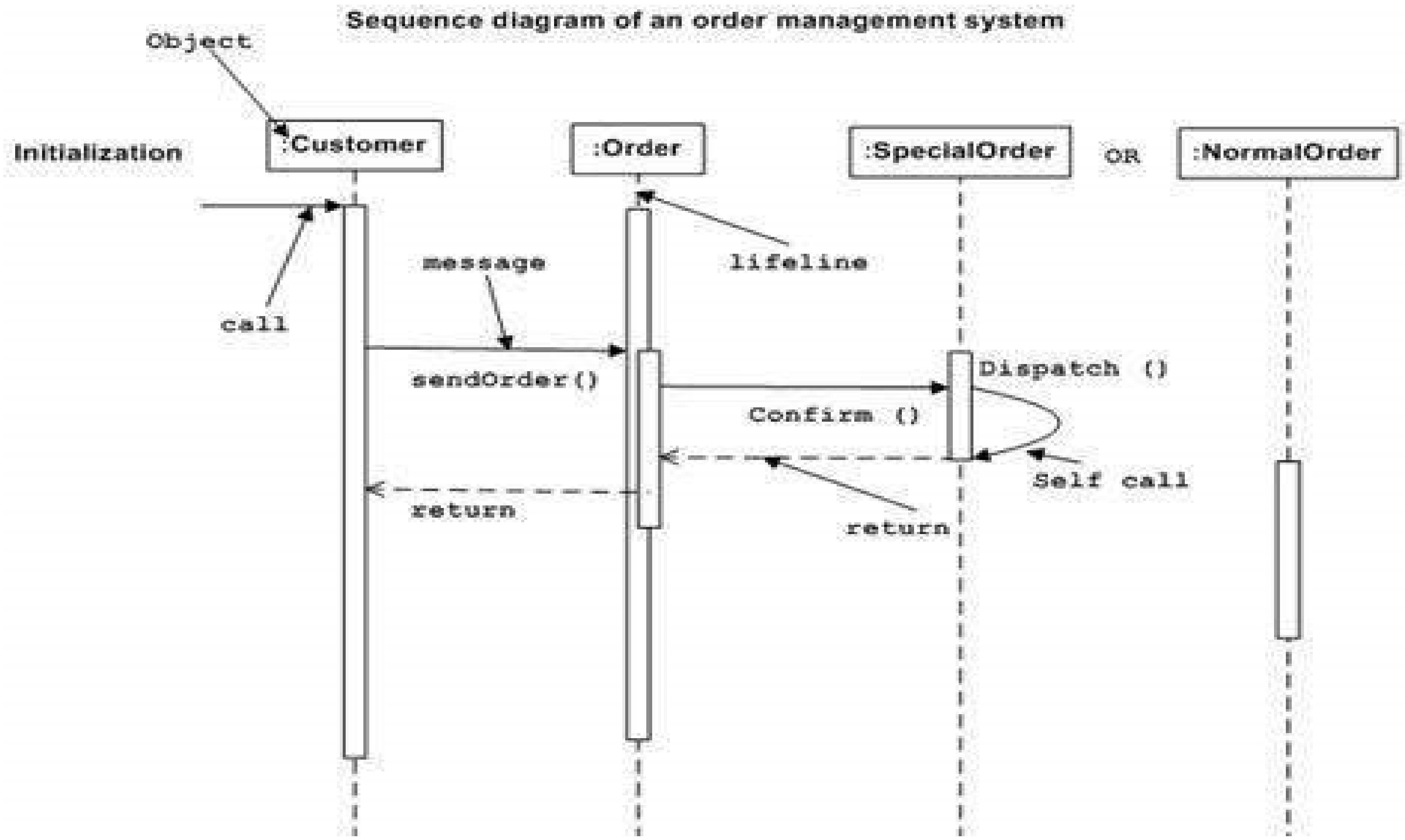


- UML diagrams generally permit the annotation of comments in all UML diagram types.
- The comment object is a rectangle with a folded-over corner.
- The comment can be linked to the related object with a dashed line.

Note: View Sequence Diagram Best Practices to learn about sequence fragments.

The Sequence Diagram

- The sequence diagram has four objects (Customer, Order, SpecialOrder and NormalOrder).
- The following diagram shows the message sequence for *SpecialOrder* object and the same can be used in case of *NormalOrder* object.
- It is important to understand the time sequence of message flows. The message flow is nothing but a method call of an object.
- The first call is *sendOrder ()* which is a method of *Order object*.
- The next call is *confirm ()* which is a method of *SpecialOrder* object and the last call is *Dispatch ()* which is a method of *SpecialOrder* object.
- The following diagram mainly describes the method calls from one object to another, and this is also the actual scenario when the system is running.

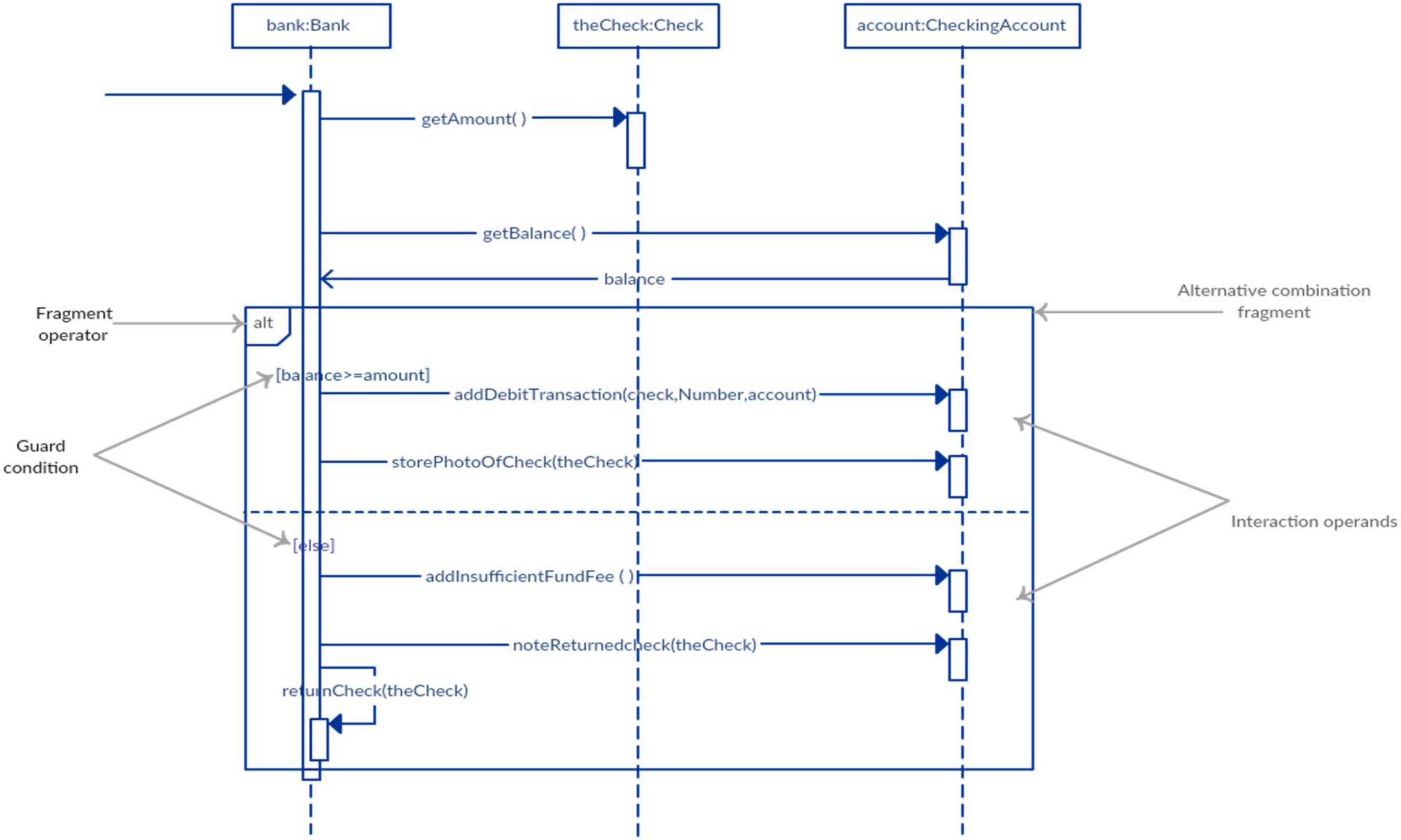


Sequence diagram of an order management system

Object
Initialization
:Customer
:Order
:SpecialOrder
OR
:NormalOrder
lifeline
message
call
sendOrder()
Dispatch ()
Confirm ()
Self call
return
return

Sequence Diagram Best Practices

- Manage complex interactions with sequence fragments
- A sequence fragment is represented as a box that frames a section of interactions between objects (as shown in the examples below) in a sequence diagram.
- It is used to show complex interactions such as alternative flows and loops in a more structured way.
- On the top left corner of the fragment sits an operator.
- This – the fragment operator – specifies what sort of a fragment it is.


bank:Bank
theCheck:Check
account:CheckingAccount
getAmount( )
getBalance( )
balance
alt
[balance>=amount]
addDebitTransaction(check,Number,account)
storePhotoOfCheck(theCheck)
[else]
addInsufficientFundFee ( )
noteReturnedcheck(theCheck)
returnCheck(theCheck)

Fragment operator

Alternative combination fragment

Guard condition

Interaction operands

Alternatives

The alternative combination fragment is used when a choice needs to be made between two or more message sequences.

It models the “if then else” logic.

The alternative fragment is represented by a large rectangle or a frame; it is specified by mentioning ‘alt’ inside the frame’s name box (a.k.a. fragment operator).

To show two or more alternatives, the larger rectangle is then divided into what is called interaction operands using a dashed line, as shown in the sequence diagram example above.

Each operand has a guard to test against and it is placed at the top left corner of the operand.

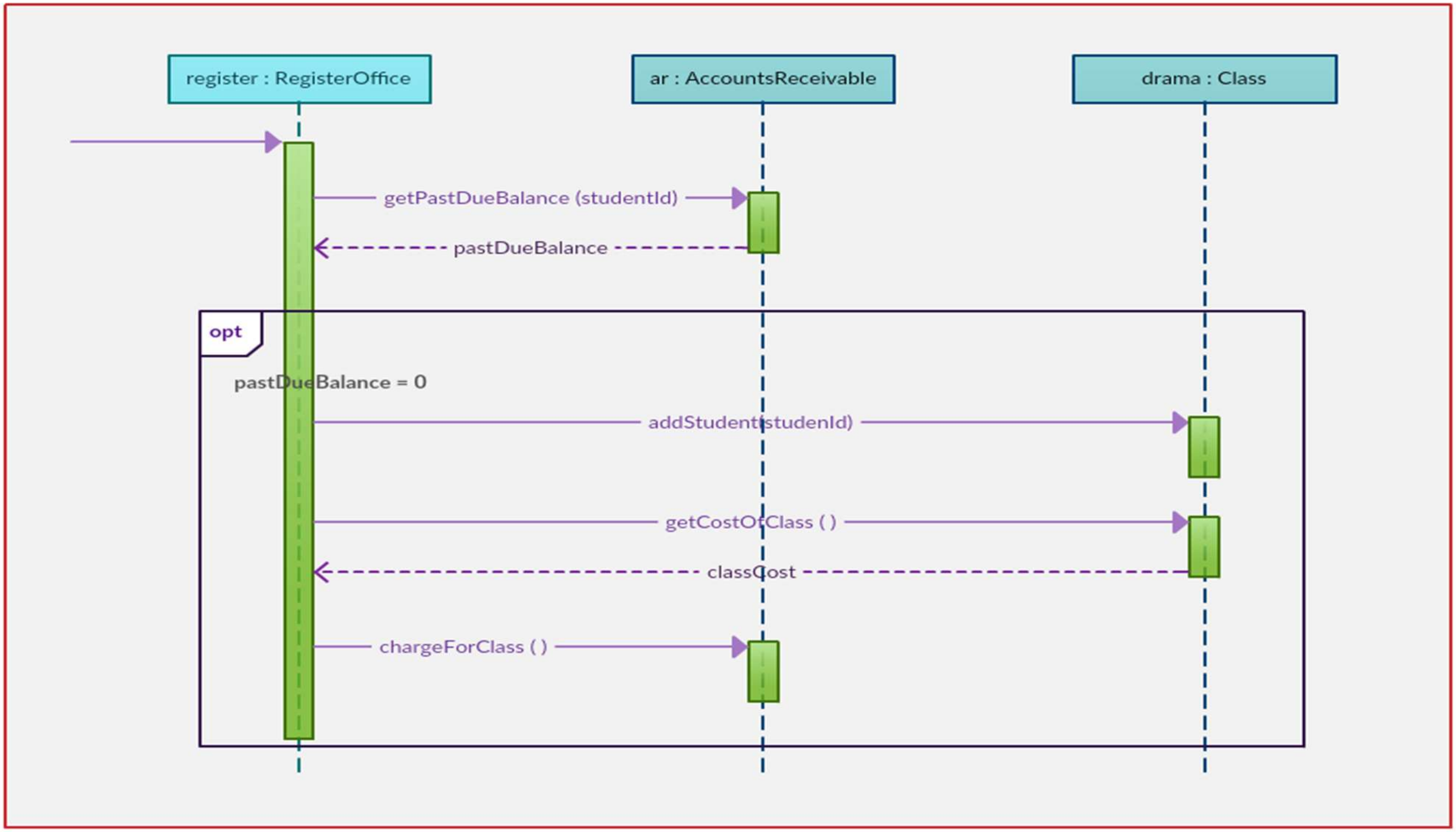
Options

The option combination fragment is used to indicate a sequence that will only occur under a certain condition, otherwise, the sequence won’t occur.

It models the “if then” statement.

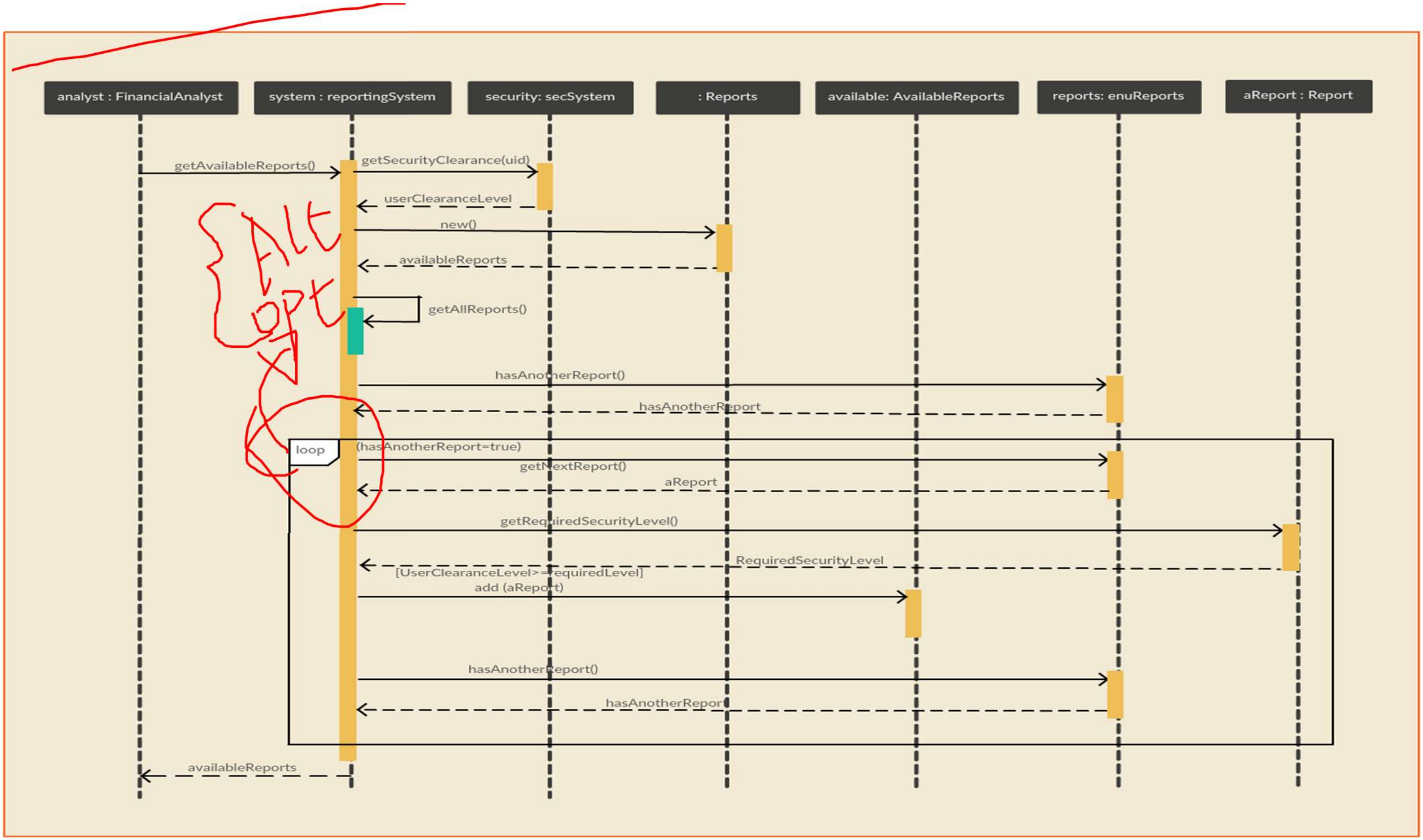
Similar to the alternative fragment, the option fragment is also represented with a rectangular frame where ‘opt’ is placed inside the name box.

Unlike the alternative fragment, an option fragment is not divided into two or more operands. Option’s guard is placed at the top left corner.


register : RegisterOffice
ar : AccountsReceivable
drama : Class
getPastDueBalance (studentId)
pastDueBalance
opt
pastDueBalance = 0
addStudent(studenId)
getCostOfClass ( )
classCost
chargeForClass ( )

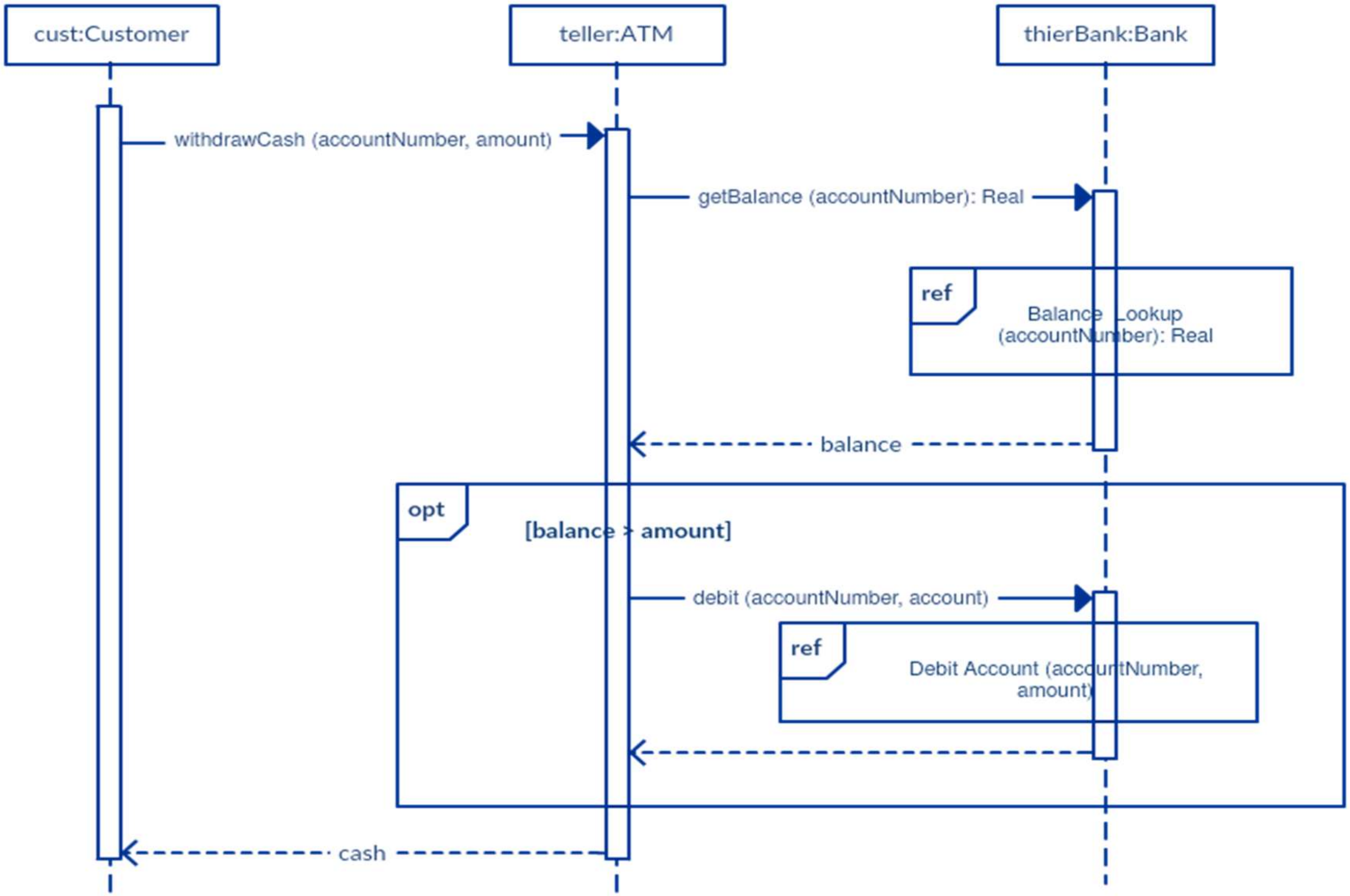
Loops

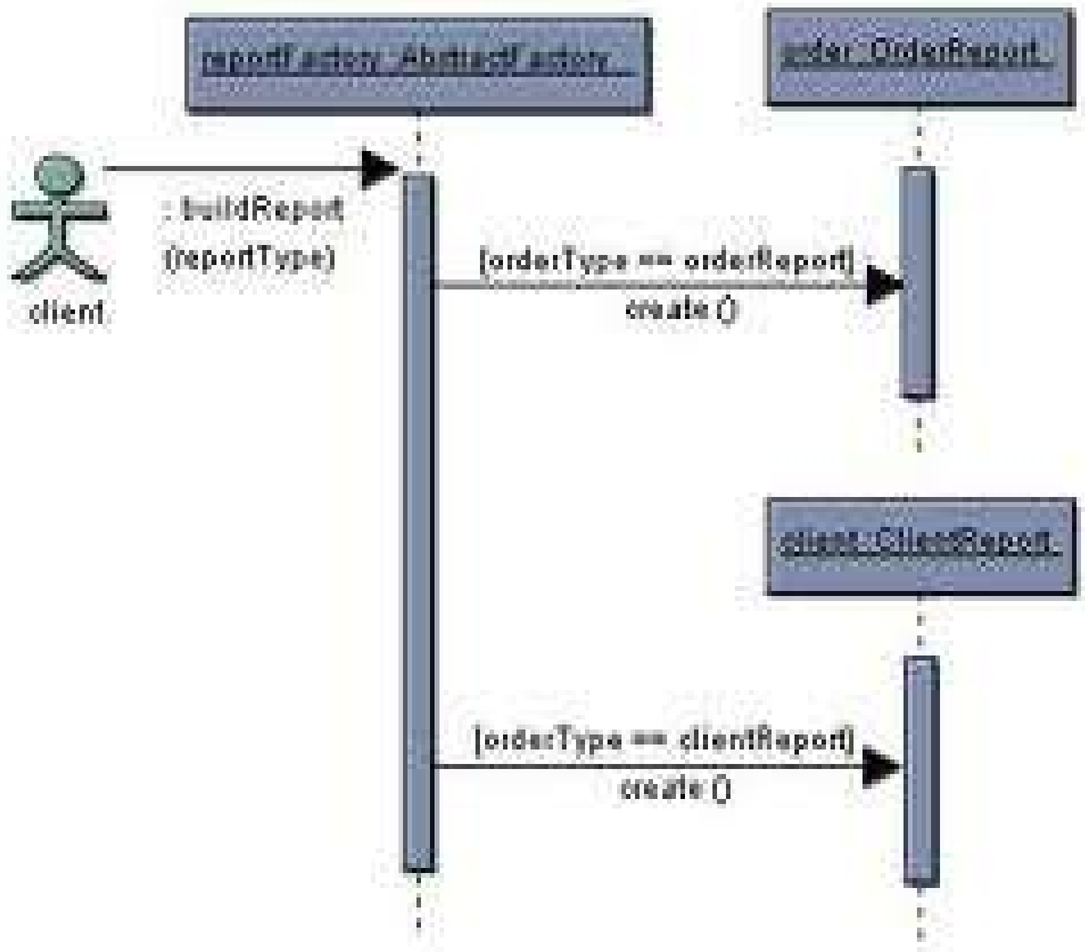
- Loop fragment is used to represent a repetitive sequence. Place the words ‘loop’ in the name box and the guard condition near the top left corner of the frame.
- In addition to the Boolean test, the guard in a loop fragment can have two other special conditions tested against.
- These are minimum iterations (written as *minint = [the number]* and maximum iterations (written as maxint = [the number]).
- If it is a minimum iterations guard, the loop must execute not less than the number mentioned, and if it is a maximum iterations guard, the loop mustn’t execute more than the number indicated.
- (Find an example of a loop fragment below in the sequence diagram templates and example section)

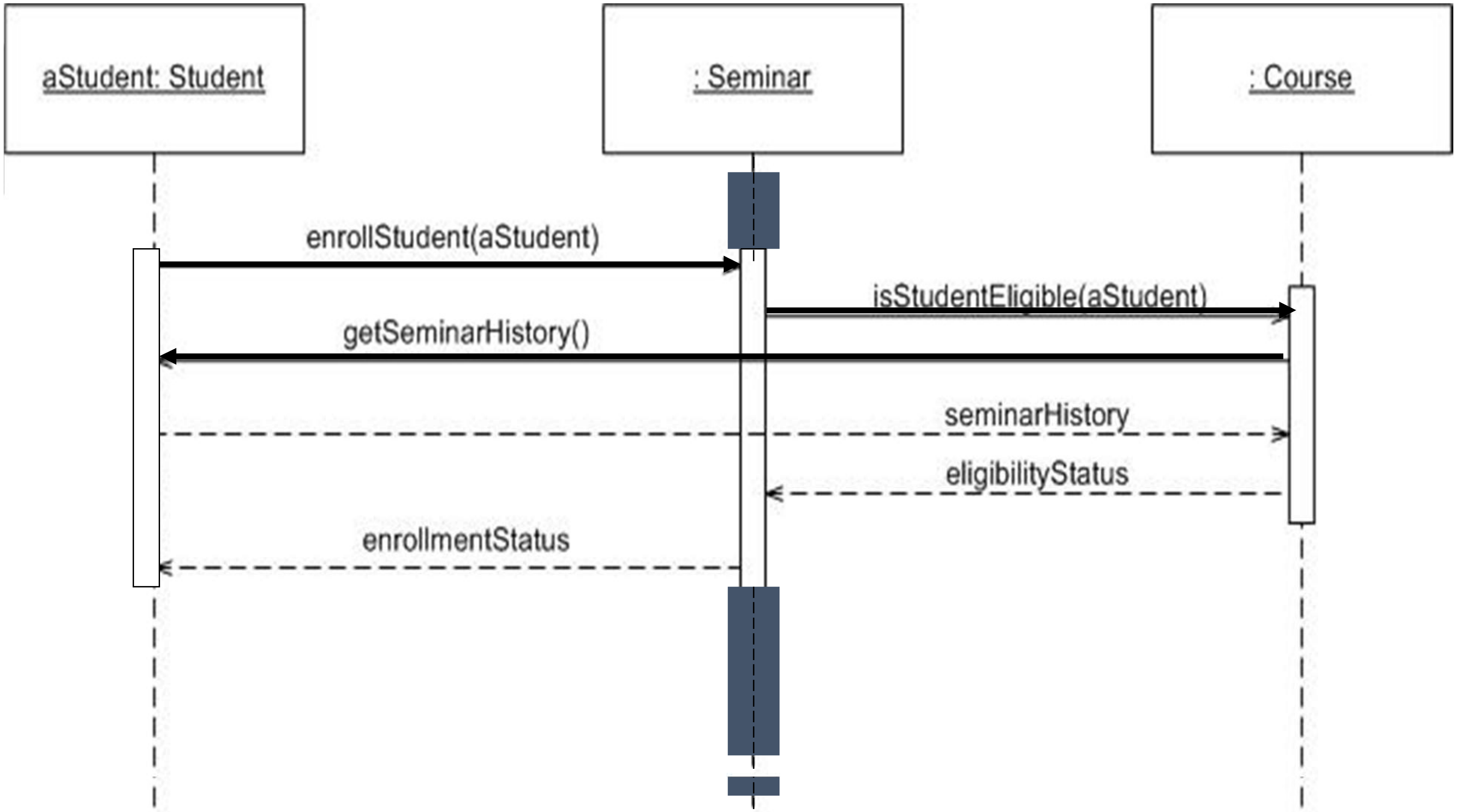

analyst : FinancialAnalyst
system : reportingSystem
security: secSystem
: Reports
available: AvailableReports
reports: enuReports
aReport : Report
getAvailableReports()
getSecurityClearance(uid)
userClearanceLevel
new()
availableReports
getAllReports()
hasAnotherReport()
hasAnotherReport
loop
(hasAnotherReport=true)
getNextReport()
aReport
getRequiredSecurityLevel()
RequiredSecurityLevel
[UserClearanceLevel>=requiredLevel]
add (aReport)
hasAnotherReport()
hasAnotherReport
availableReports
Alt
opt

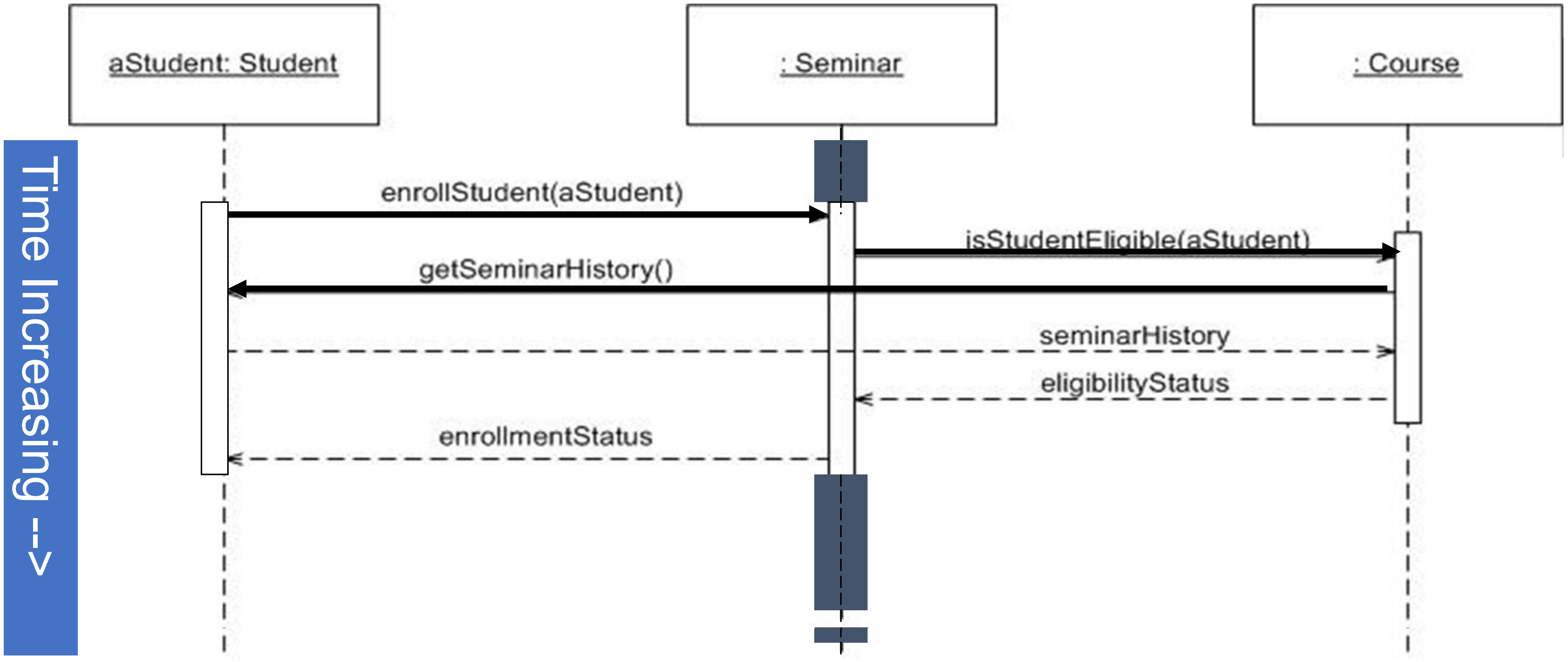
Reference Fragment

- You can use the ref fragment to manage the size of large sequence diagrams.
- It allows you to reuse part of one sequence diagram in another, or in other words, you can reference part of a diagram in another diagram using the ref fragment.
- To specify the reference fragment, you have to mention ‘ref’ in the name box of the frame and the name of the sequence diagram that is being referred to inside the frame.


cust:Customer
teller:ATM
thierBank:Bank
withdrawCash (accountNumber, amount)
getBalance (accountNumber): Real
ref
Balance Lookup (accountNumber): Real
balance
opt
[balance > amount]
debit (accountNumber, account)
ref
Debit Account (accountNumber, amount)
cash


reportFactory :AbstractFactory
order :OrderReport
client
: buildReport (reportType)
[orderType == orderReport] create ()
client :ClientReport
[orderType == clientReport] create ()

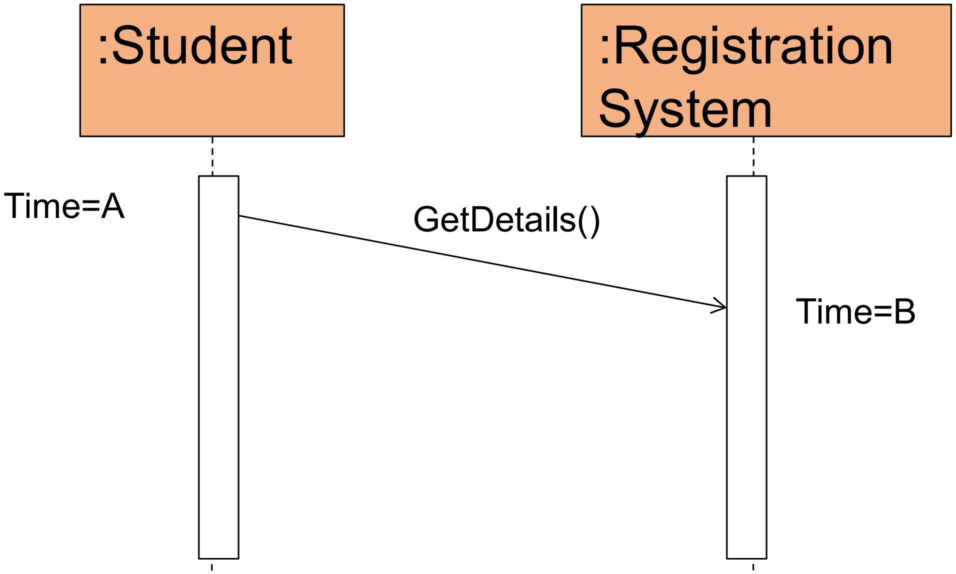

aStudent: Student
: Seminar
: Course
enrollStudent(aStudent)
isStudentEligible(aStudent)
getSeminarHistory()
seminarHistory
eligibilityStatus
enrollmentStatus


aStudent: Student
: Seminar
: Course
Time Increasing -->
enrollStudent(aStudent)
isStudentEligible(aStudent)
getSeminarHistory()
seminarHistory
eligibilityStatus
enrollmentStatus

Lower = Later!

All lines should be horizontal to indicate instantaneous actions. Additionally if ActivityA happens before ActivityB, ActivityA must be above ActivityB

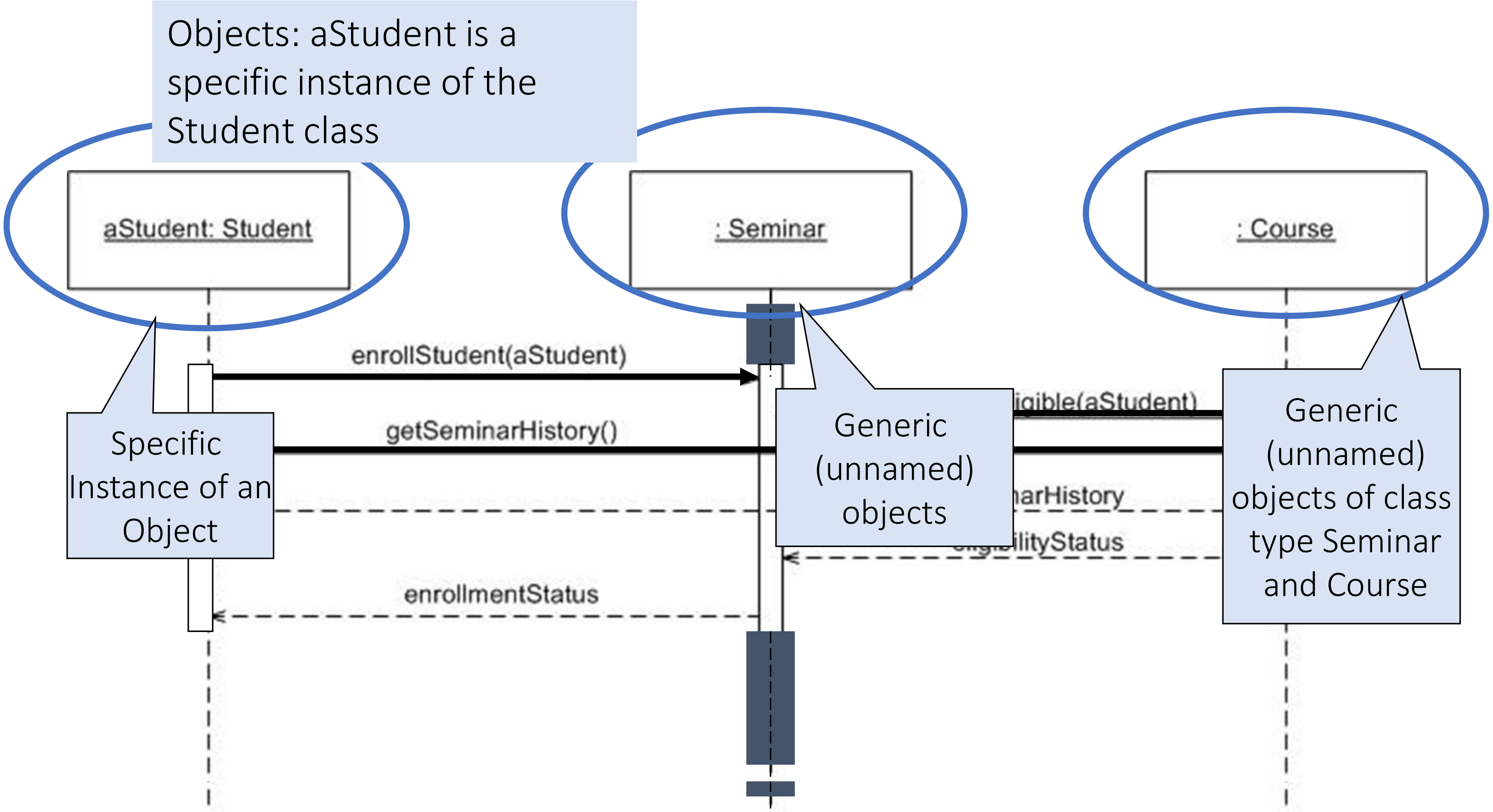
Diagonal Lines


:Student
:Registration System
Time=A
GetDetails()
Time=B

- What does this mean?

Do you typically care?

Components


aStudent: Student
: Seminar
: Course
enrollStudent(aStudent)
igible(aStudent)
getSeminarHistory()
arHistory
ilityStatus
enrollmentStatus

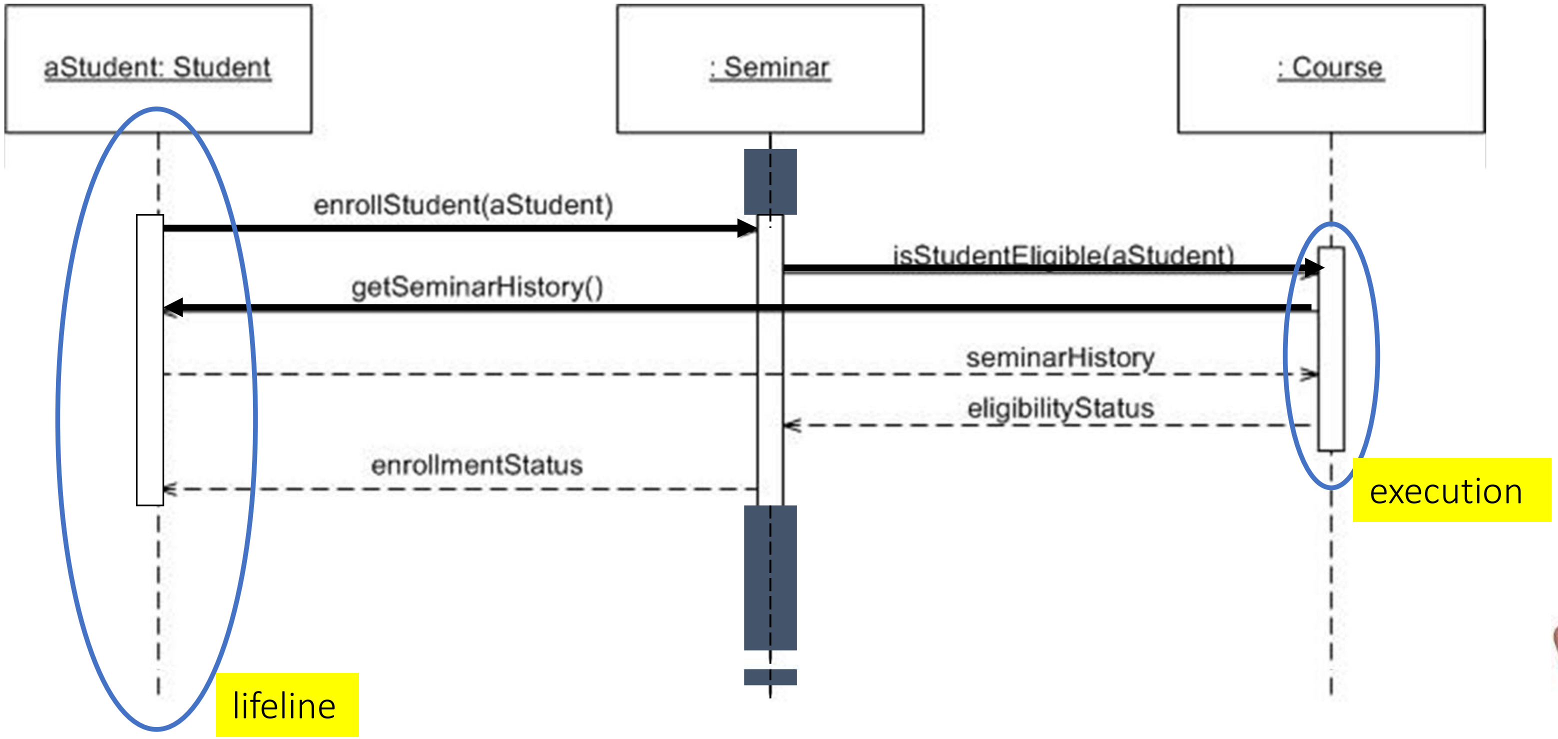
Objects: aStudent is a specific instance of the Student class

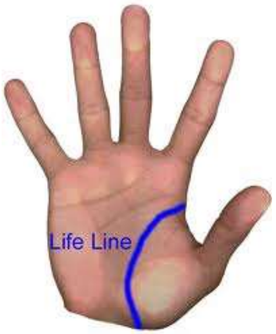
Specific Instance of an Object

Generic (unnamed) objects

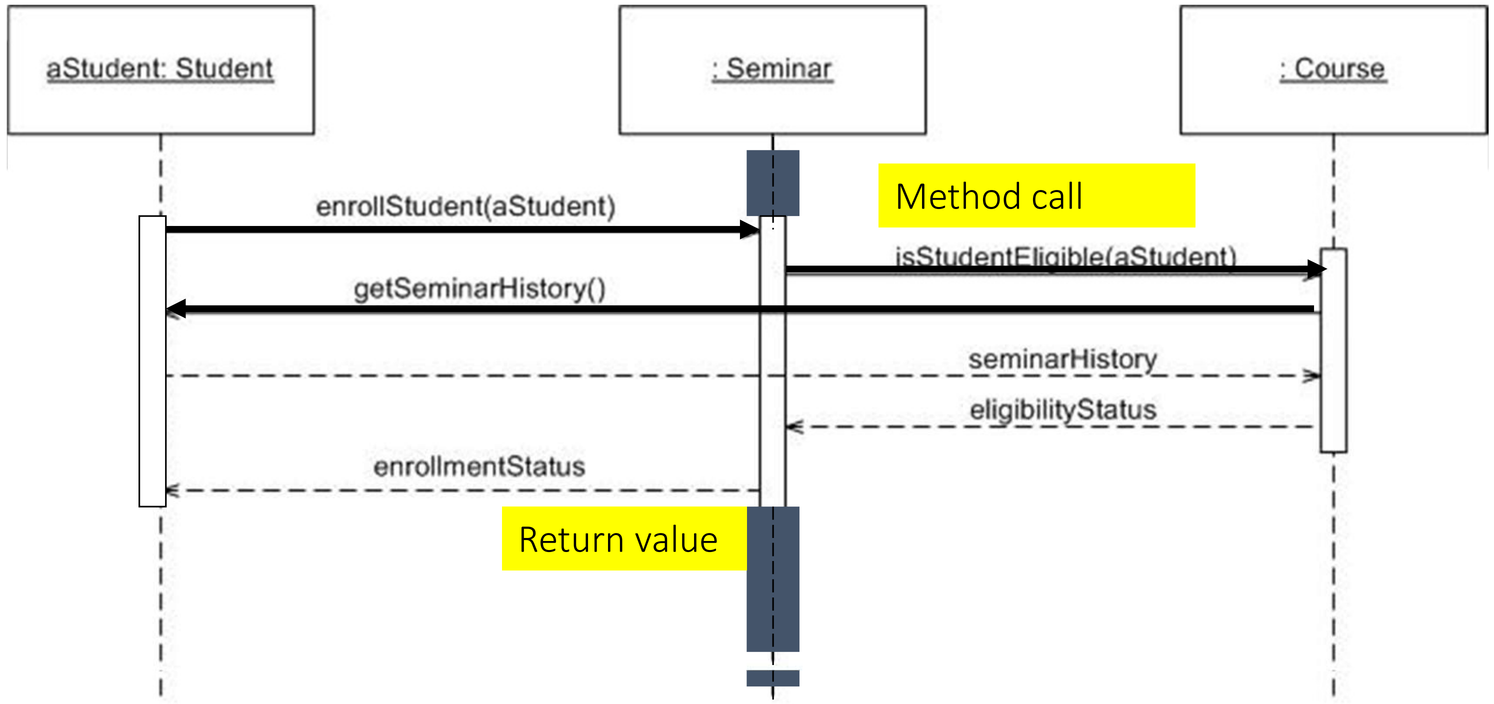
Generic (unnamed) objects of class type Seminar and Course

Components


aStudent: Student
: Seminar
: Course
enrollStudent(aStudent)
isStudentEligible(aStudent)
getSeminarHistory()
seminarHistory
eligibilityStatus
enrollmentStatus
execution
lifeline


Life Line

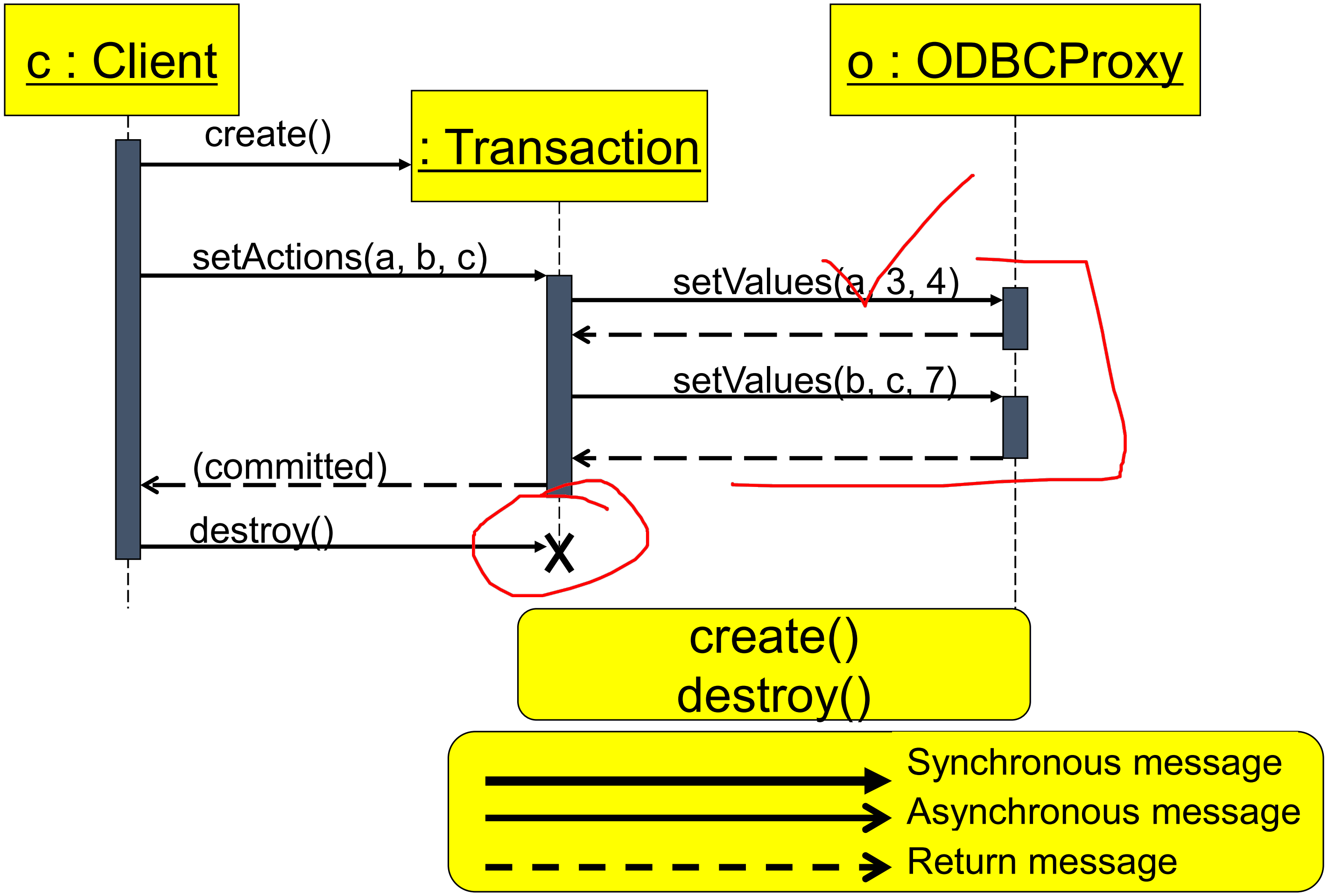
Components


aStudent: Student
: Seminar
: Course
enrollStudent(aStudent)
isStudentEligible(aStudent)
getSeminarHistory()
seminarHistory
eligibilityStatus
enrollmentStatus

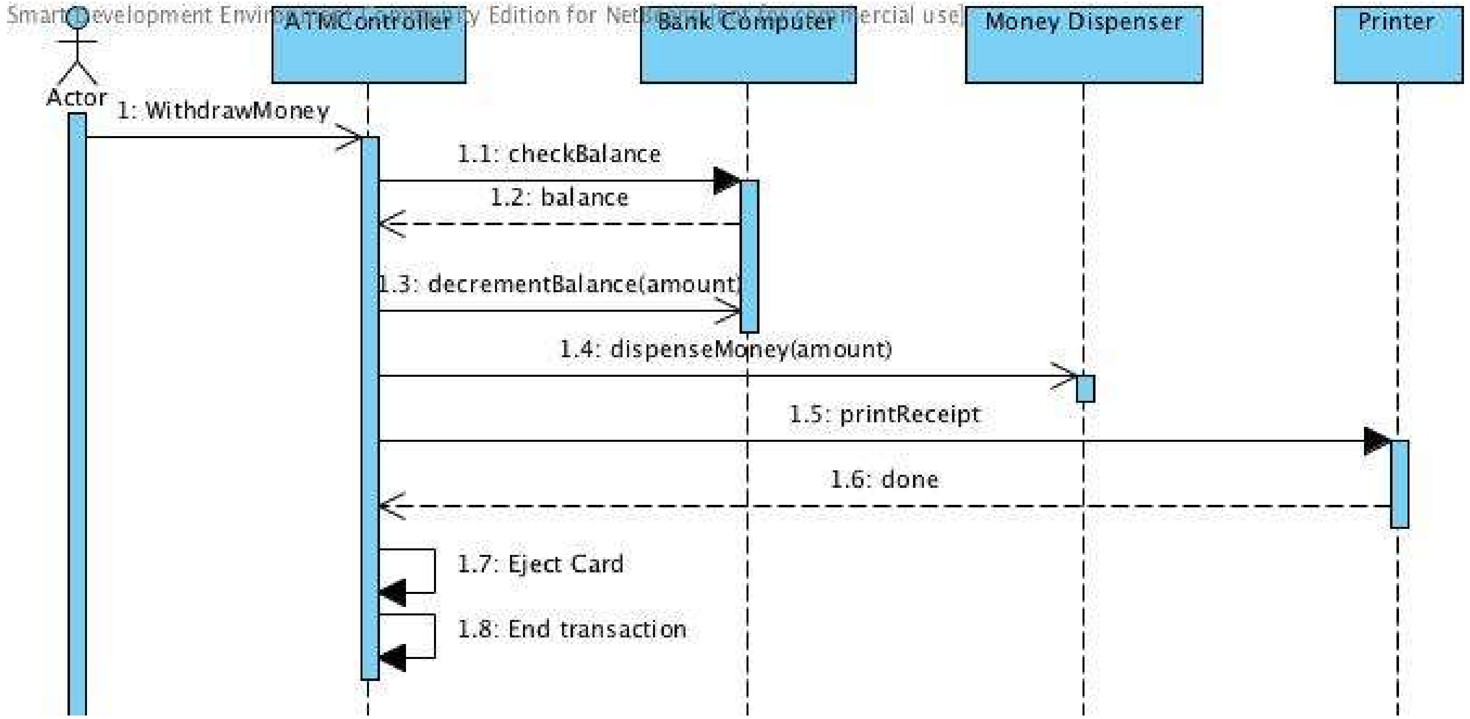
Method call

Return value

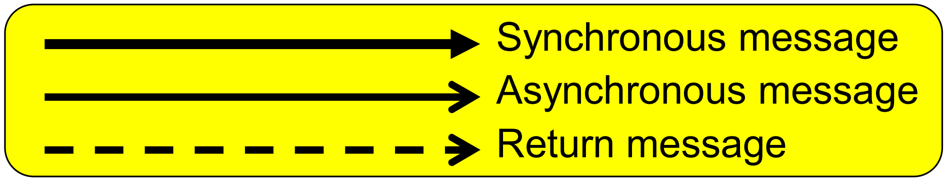
Components


c : Client
: Transaction
o : ODBCProxy
create()
setActions(a, b, c)
setValues(a, 3, 4)
setValues(b, c, 7)
(committed)
destroy()
create()
destroy()
Synchronous message
Asynchronous message
Return message

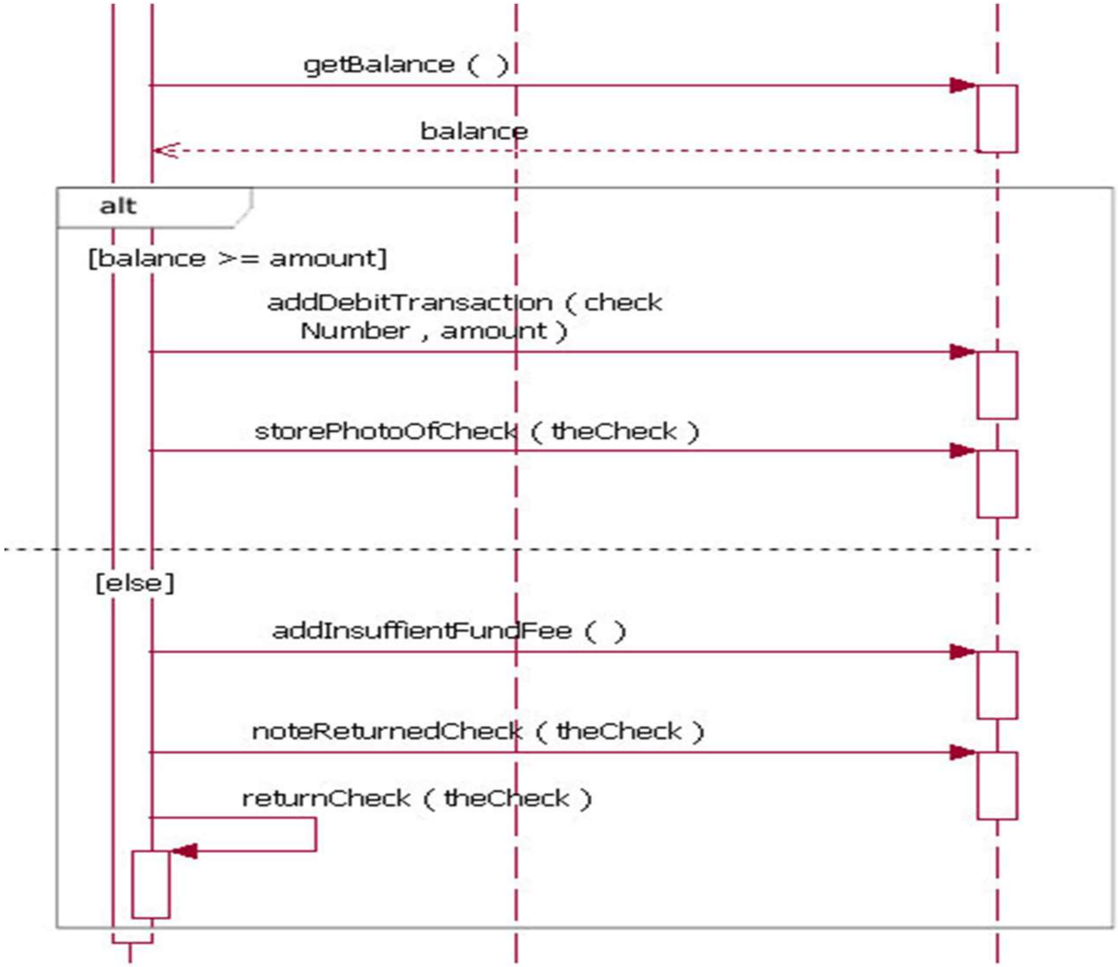
Async Message Example


Actor
ATMController
Bank Computer
Money Dispenser
Printer
1: WithdrawMoney
1.1: checkBalance
1.2: balance
1.3: decrementBalance(amount)
1.4: dispenseMoney(amount)
1.5: printReceipt
1.6: done
1.7: Eject Card
1.8: End transaction

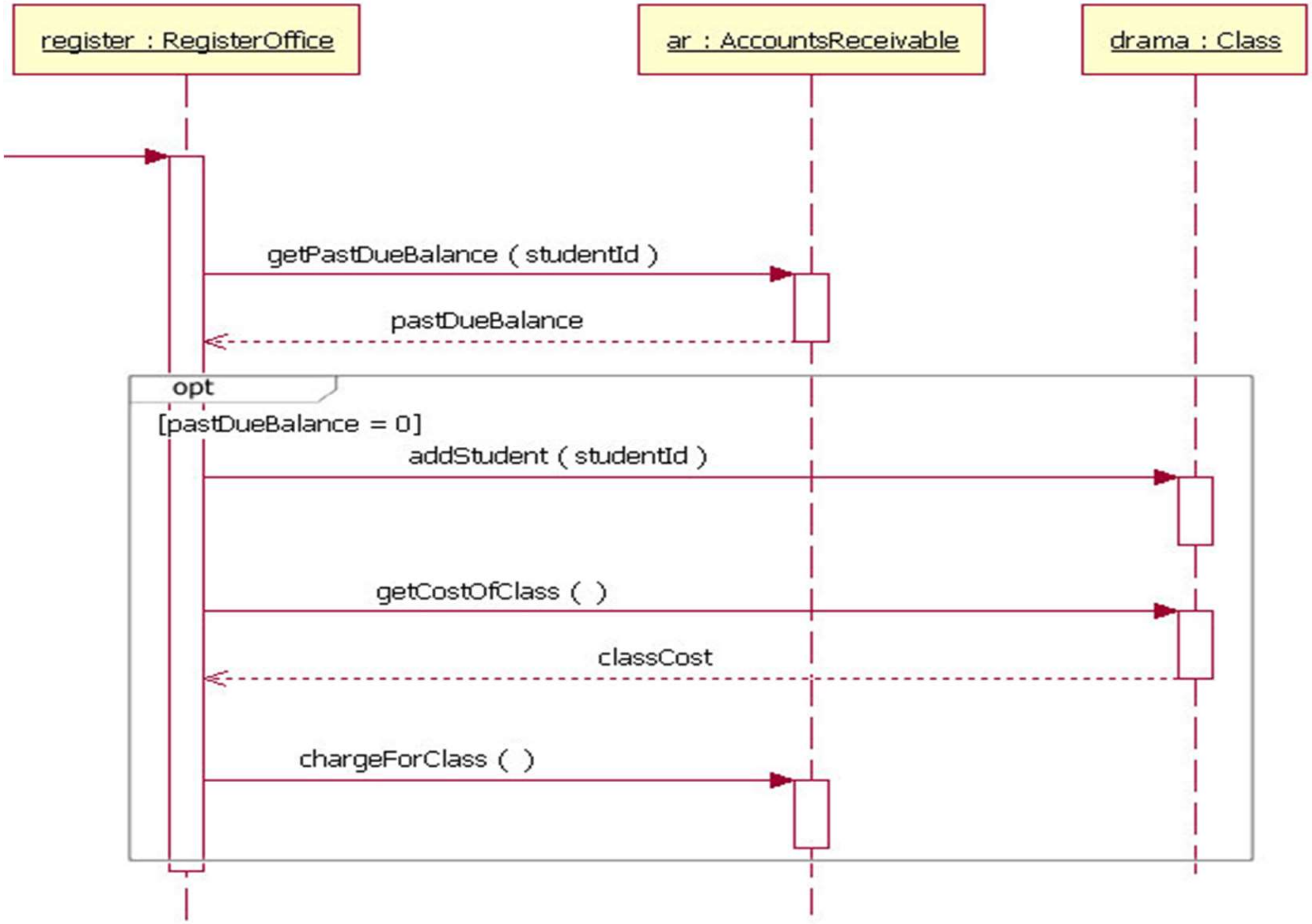
There are problems here… what are they?


Synchronous message
Asynchronous message
Return message

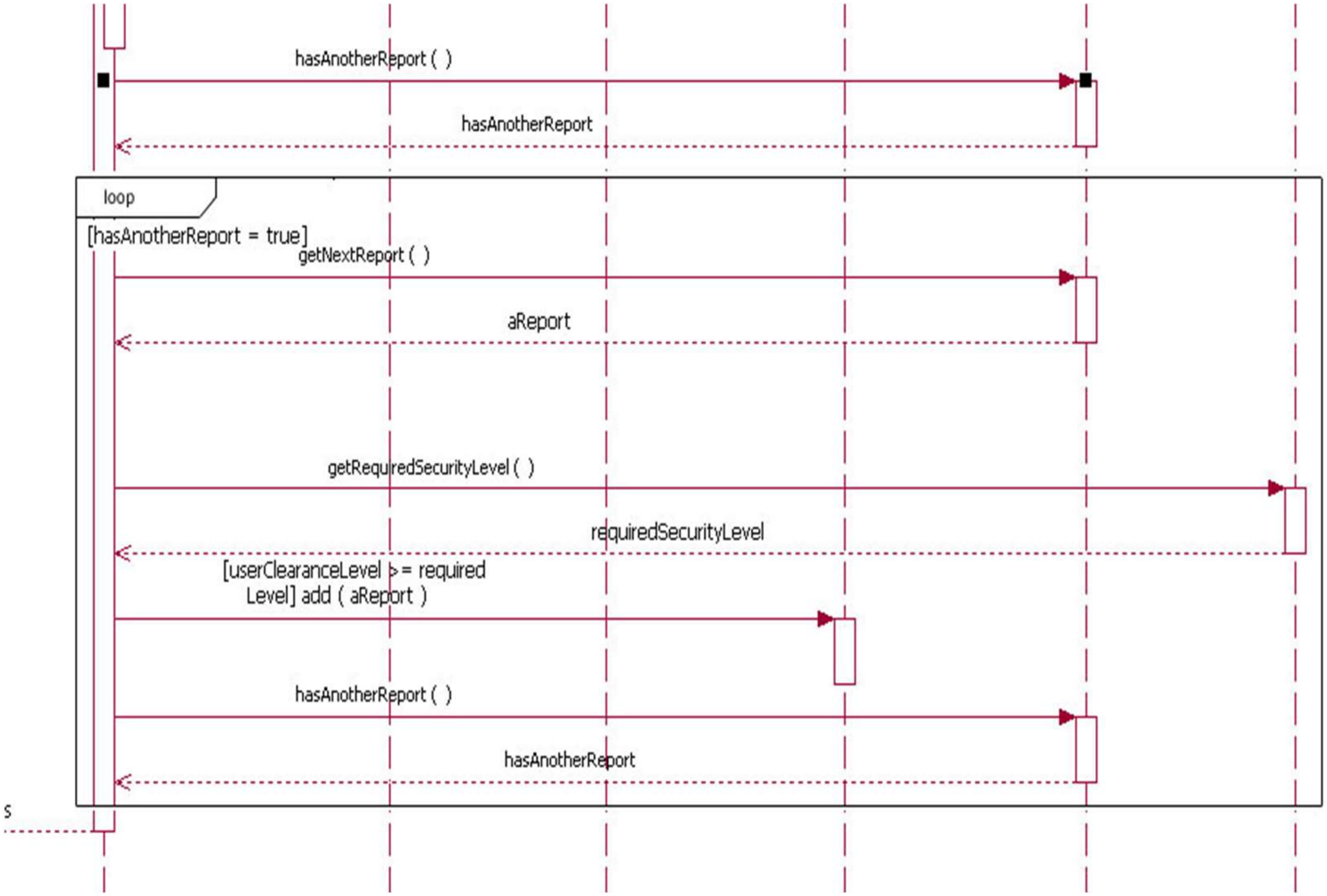
Components: alt/else


getBalance ( )
balance
alt
[balance >= amount]
addDebitTransaction (check Number , amount)
storePhotoOfCheck (theCheck)
[else]
addInsuffientFundFee ( )
noteReturnedCheck (theCheck)
returnCheck (theCheck)

Components: option


register : RegisterOffice
ar : AccountsReceivable
drama : Class
getPastDueBalance (studentId)
pastDueBalance
opt
[pastDueBalance = 0]
addStudent (studentId)
getCostOfClass ( )
classCost
chargeForClass ( )

Components: loop

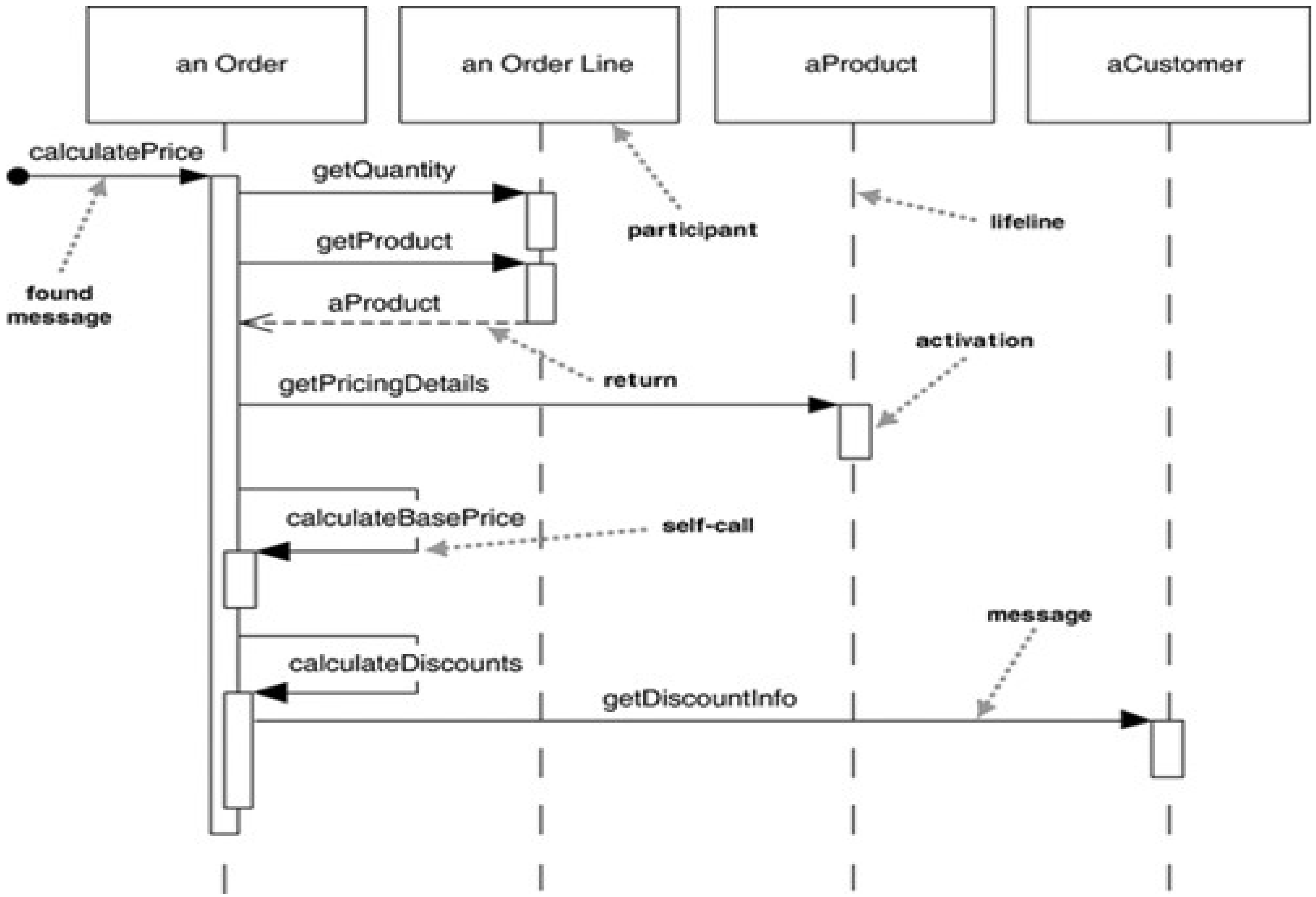

hasAnotherReport ( )
hasAnotherReport
loop
[hasAnotherReport = true]
getNextReport ( )
aReport
getRequiredSecurityLevel ( )
requiredSecurityLevel
[userClearanceLevel >= required Level] add (aReport)
hasAnotherReport ( )
hasAnotherReport
s

Rules of thumb

- Rarely use options, loops, alt/else
  - These constructs complicate a diagram and make them hard to read/interpret.
  - Frequently it is better to create multiple simple diagrams
- Create sequence diagrams for use cases when it helps clarify and visualize a complex flow
- Remember: the goal of UML is communication and understanding

Sequence Diagrams

- Interaction diagrams describe how groups of objects collaborate in some behavior
- The UML defines several forms of interaction diagram, the most common is the sequence diagram
- A class diagram shows a fixed view of a system
- A sequence diagram represents a dynamic view of a system by capturing message sends over time
- Can document a scenario such as
- Dealer deals cards to all players
- Withdraw Money when there is enough balance


an Order
an Order Line
aProduct
aCustomer
calculatePrice
found message
getQuantity
participant
lifeline
getProduct
aProduct
return
activation
getPricingDetails
calculateBasePrice
self-call
message
calculateDiscounts
getDiscountInfo

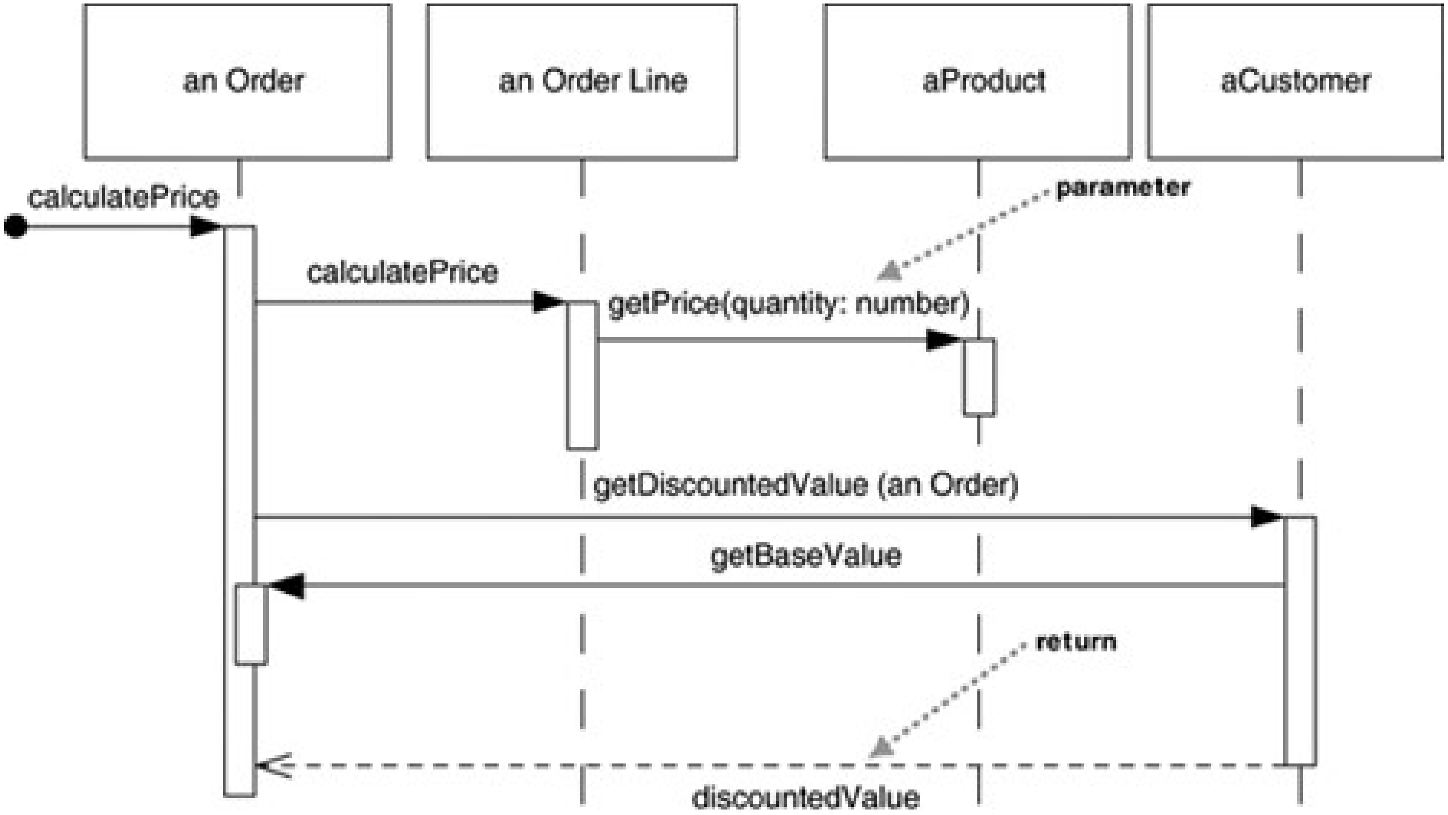
Sequence Diagrams

- Not good at showing details of algorithms such as loops and conditional

- Good at showing the calls between participants

- Gives a good picture about which participants are doing which processing

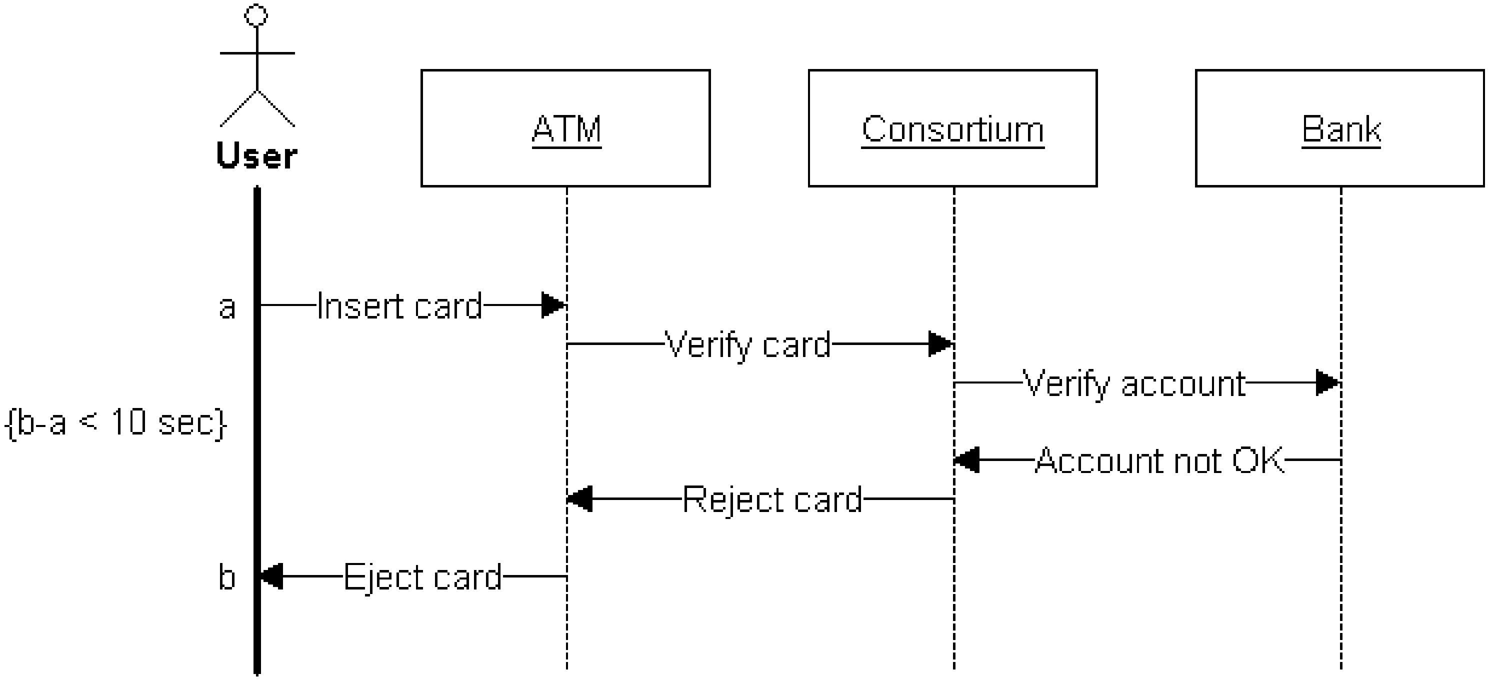
More Distributed Control Shown here


an Order
an Order Line
aProduct
aCustomer
calculatePrice
parameter
calculatePrice
getPrice(quantity: number)
getDiscountedValue (an Order)
getBaseValue
return
discountedValue

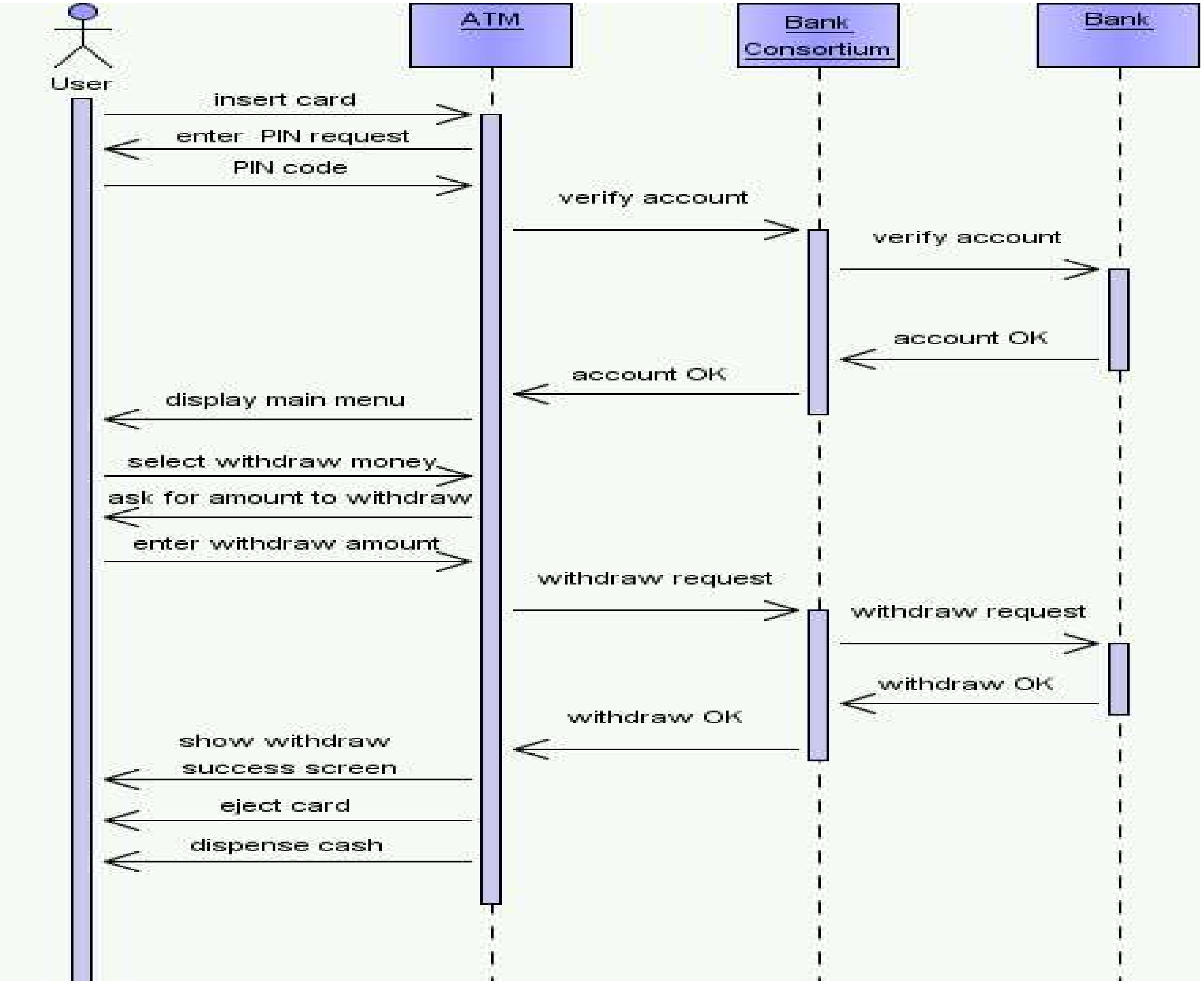
Syntax

- Objects are lined up on top in rectangles
- Object names : CardReader
- Dashed lines represent lifetime of objects
- Rectangles are activation lines
- When the object is "alive“
- Activation bar of the receivers of the message is smaller than the sender's activation bar
- Not much detail written

Another Example

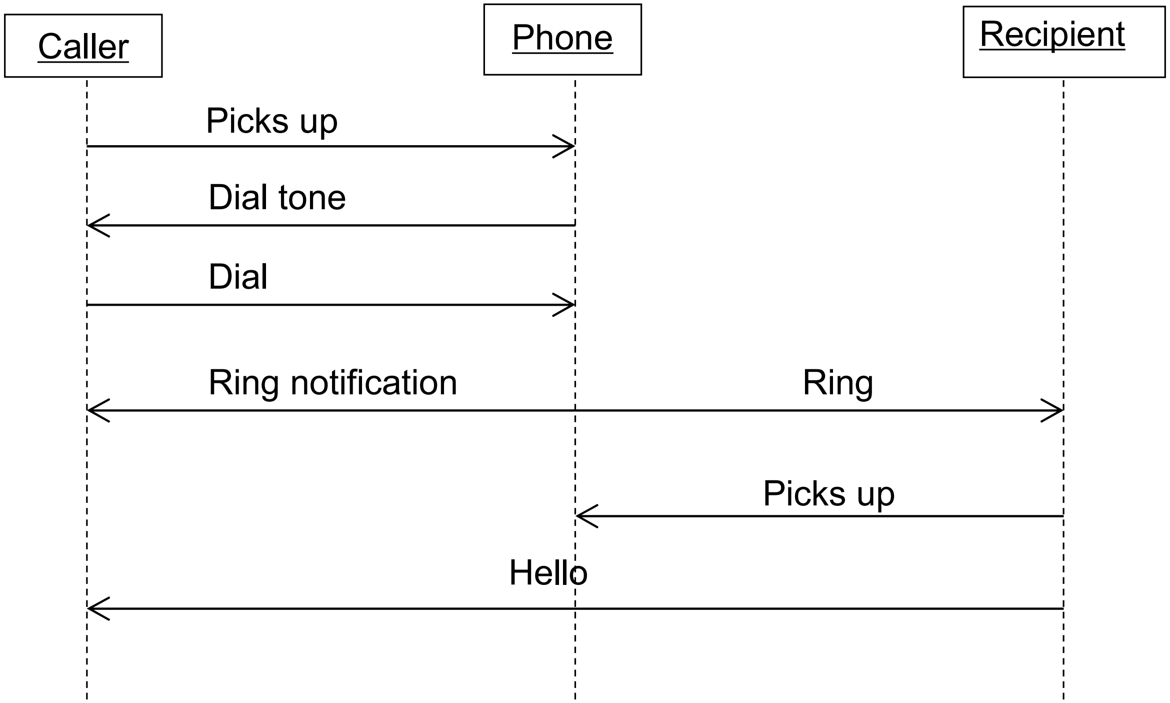

User
ATM
Consortium
Bank
a
Insert card
Verify card
Verify account
{b-a < 10 sec}
Account not OK
Reject card
b
Eject card

Scenario: The user tries to use an ATM, but the account is not known


User
ATM
Bank Consortium
Bank
insert card
enter PIN request
PIN code
verify account
verify account
account OK
account OK
display main menu
select withdraw money
ask for amount to withdraw
enter withdraw amount
withdraw request
withdraw request
withdraw OK
withdraw OK
show withdraw success screen
eject card
dispense cash

Scenario: The user successfully withdraws money from an ATM

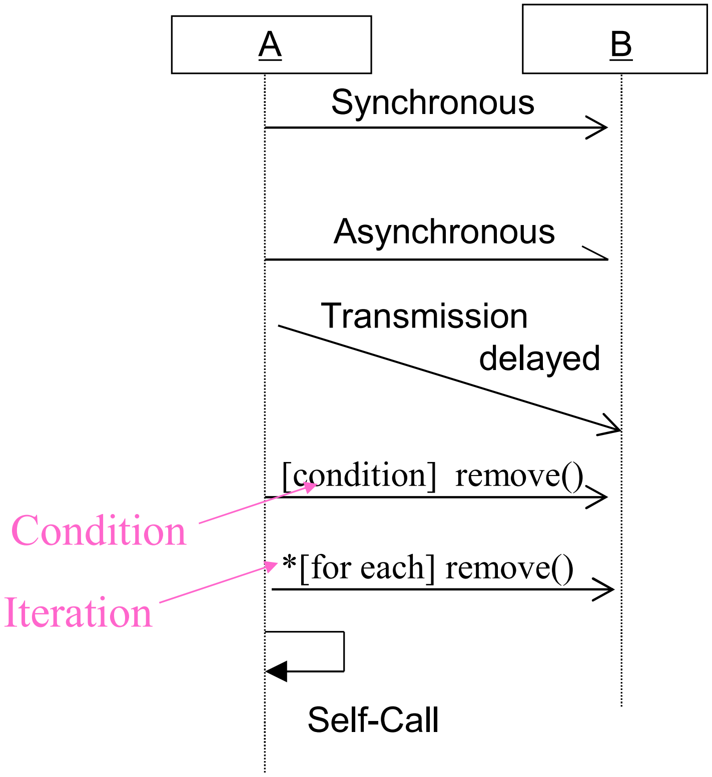
Sequence Diagram(make a phone call)


Caller
Phone
Recipient
Picks up
Dial tone
Dial
Ring notification
Ring
Picks up
Hello

Sequence Diagram: Object interaction

Self-Call: A message that an Object sends to itself.

Condition: indicates when a message is sent. The message is sent only if the condition is true.

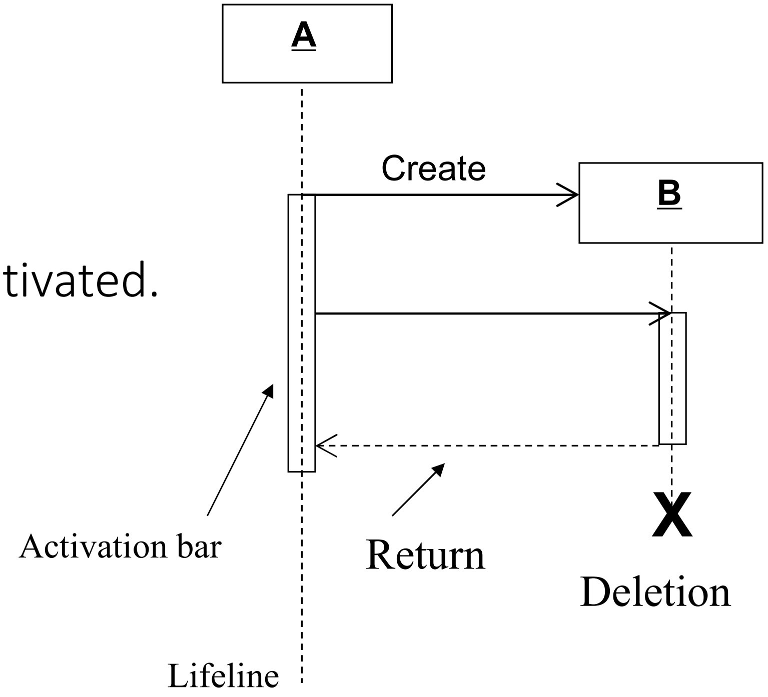

A
B
Synchronous
Asynchronous
Transmission
delayed
[condition] remove()
Condition
*[for each] remove()
Iteration
Self-Call

Sequence Diagrams – Object Life Spans

- Creation
  - Create message
  - Object life starts at that point

- Activation
  - Symbolized by rectangular stripes
  - Place on the lifeline where object is activated.
  - Rectangle also denotes when object is deactivated.

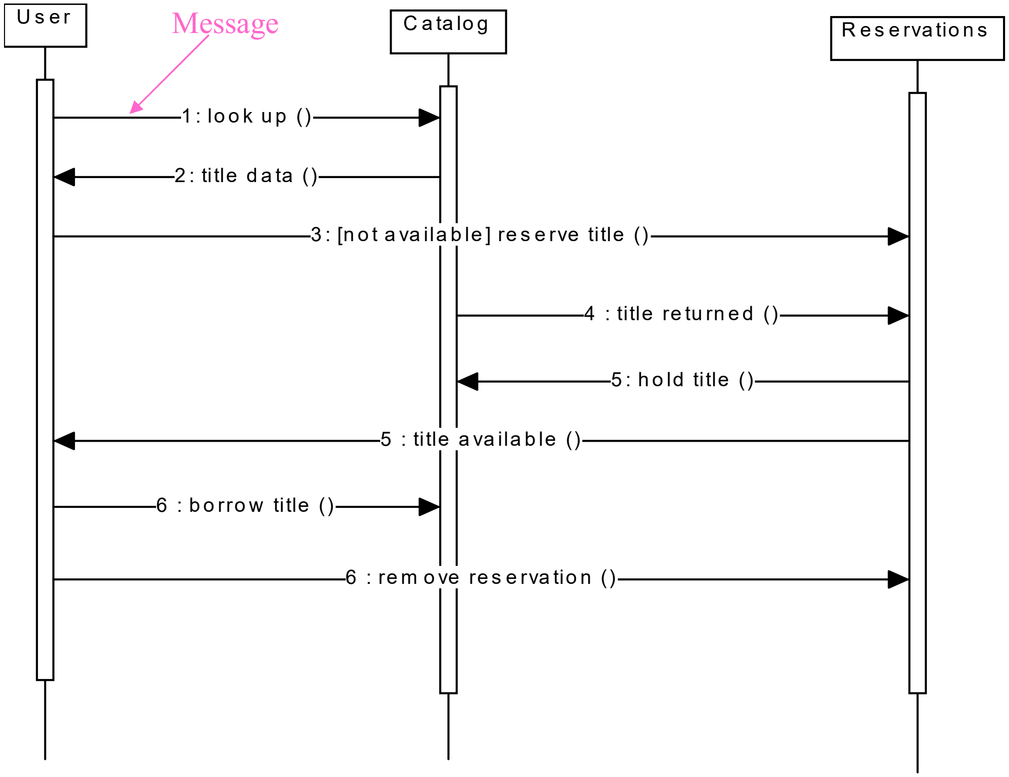
- Deletion
  - Placing an ‘X’ on lifeline
  - Object’s life ends at that point

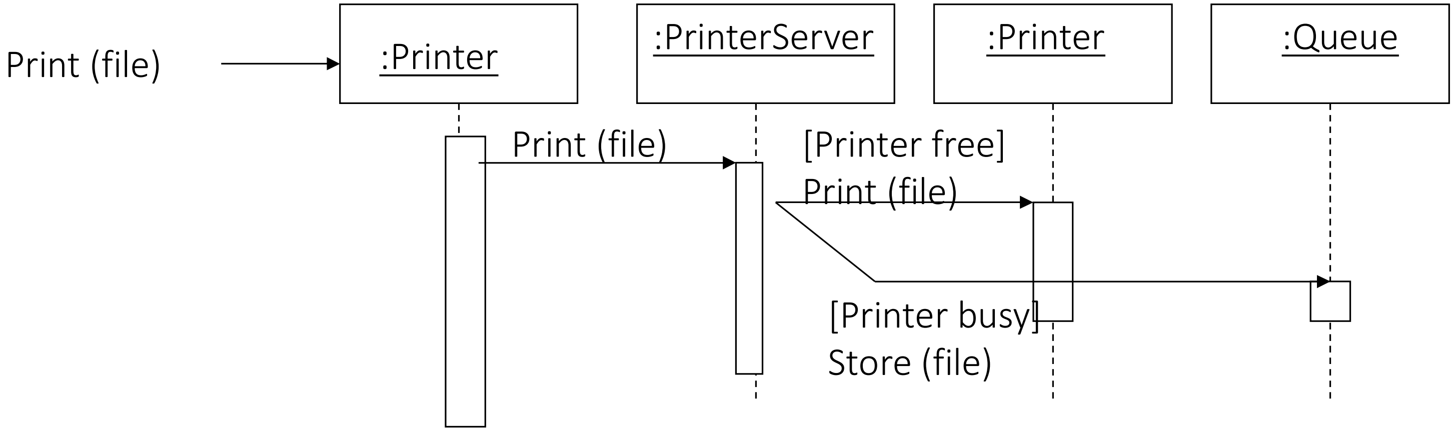

A
Create
B
X
Activation bar
Return
Deletion
Lifeline

Sequence Diagram

- Sequence diagrams demonstrate the behavior of objects in a use case by describing the objects and the messages they pass.

- The horizontal dimension shows the objects participating in the interaction.
- The vertical arrangement of messages indicates their order.
- The labels may contain the seq. # to indicate concurrency.

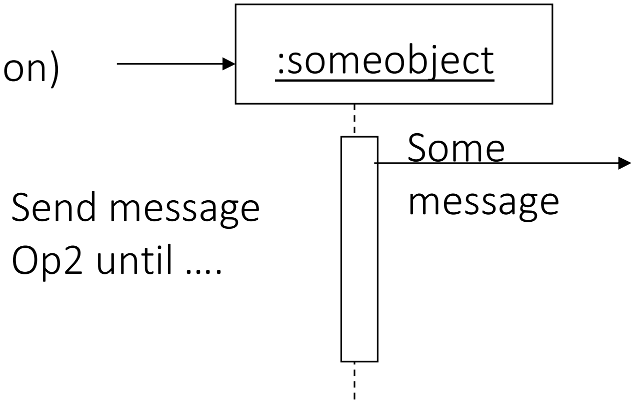

User
Message
Catalog
Reservations
1 : look up ()
2 : title data ()
3 : [not available] reserve title ()
4 : title returned ()
5 : hold title ()
5 : title available ()
6 : borrow title ()
6 : remove reservation ()


Print (file)
:Printer
:PrinterServer
:Printer
:Queue
Print (file)
[Printer free]
Print (file)
[Printer busy]
Store (file)

The items in brackets are called guards.

If true, proceed; if not, don't. Just like the IFTHENELSE>

Can also say: (iteration)

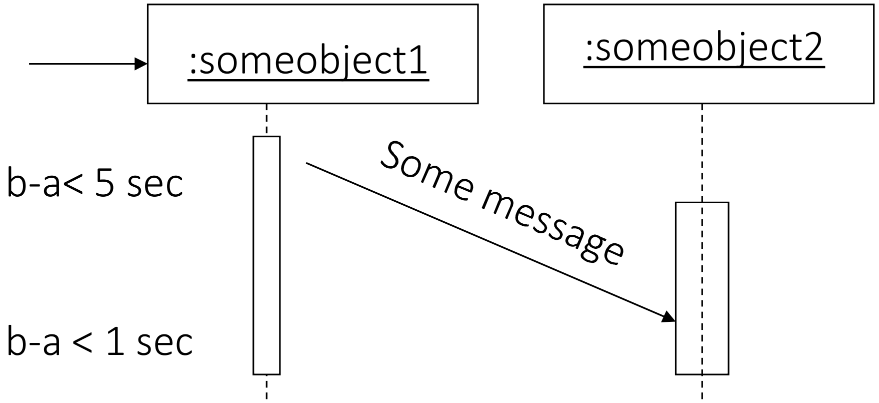


:someobject

Send message
Op2 until ….

Some
message

Can also say: (performance message…)



:someobject1

:someobject2

b-a< 5 sec

Some message

b-a < 1 sec

More information on Sequence Diagrams.

- Sequence diagrams can be used in two forms: the generic form and the instance form.
- The instance form describes a specific scenario in detail; it documents one possible interaction.
- The instance form does not have any conditions, branches, or loops; it shows the interaction for just the chosen scenario.
- The generic form describes all possible alternatives in a scenario; therefore, branches, loops, conditions may be identified.

More information on Sequence Diagrams.

- For example, the scenario “opening an account” in a sequence diagram using the generic form would be described with all possible alternatives; where money is immediately deposited in the account and so on.

- The same scenario documented with the instance form of a sequence diagram would have to choose one specific execution and stick to that case; for example, one diagram could show the successful opening of an account.

- If all cases must be shown using instance form diagrams, a number of them would have to be drawn.

- Here is one way to indicate a selection…

Sequence Diagram Common Mistakes

- When drawing sequence diagrams, designers tend to make these common mistakes.
- By avoiding these mistakes you can ensure the quality of your diagram.
- Adding too much detail.
- This clutters up the diagram and makes it difficult to read.
- Obsolete and out of date sequence diagrams that are irrelevant when compared to the interfaces, actual architectures etc. of the system.
- Don’t forget to replace them or modify them.
- Leaving no blank space between the use case text and the message arrow; this makes it difficult for anyone to read the diagram.
- Not considering the origins of message arrows carefully.

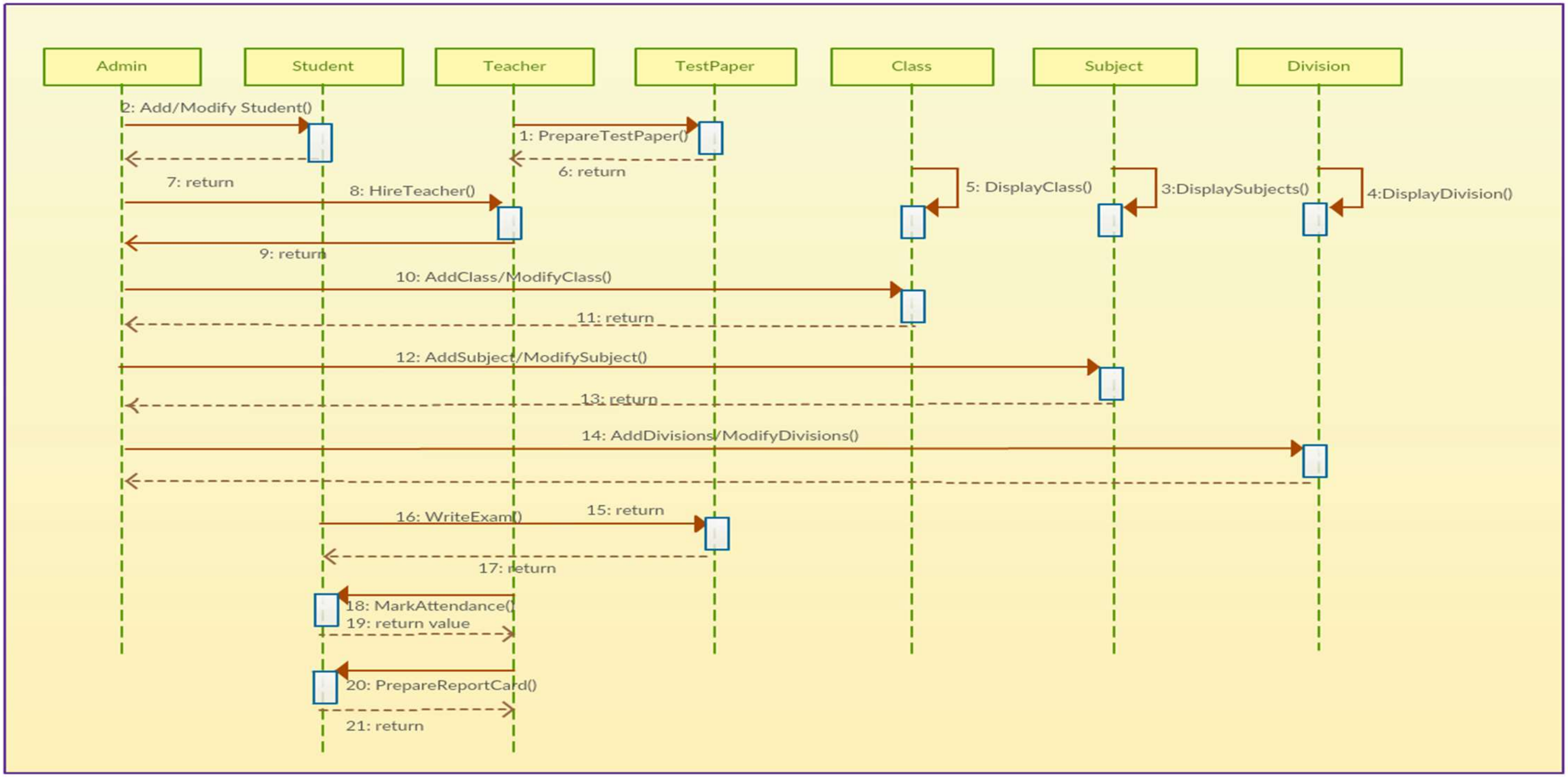
Summary

- Sequence diagrams model object interactions with an emphasis on time ordering
- Method call lines
  - Must be horizontal!

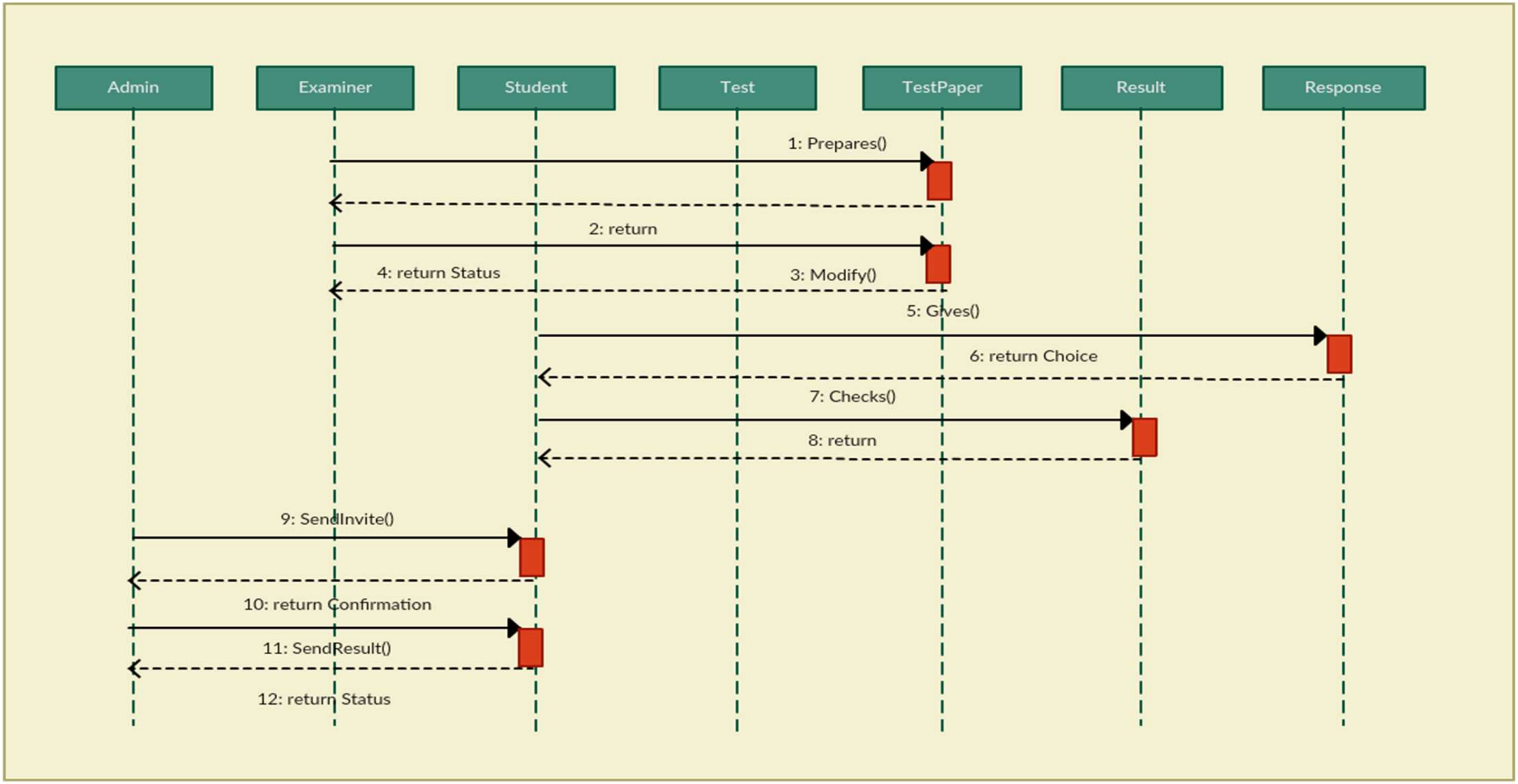
  - Vertical height matters!

    "Lower equals Later"
  - Label the lines
- Lifeline – dotted vertical line
- Execution bar – bar around lifeline when code is running
- Arrows
  - Synchronous call (you're waiting for a return value) – triangle arrow-head
  - Asynchronous call (not waiting for a return) – open arrow-head
  - Return call – dashed line

Sequence Diagram Example of a School Management System

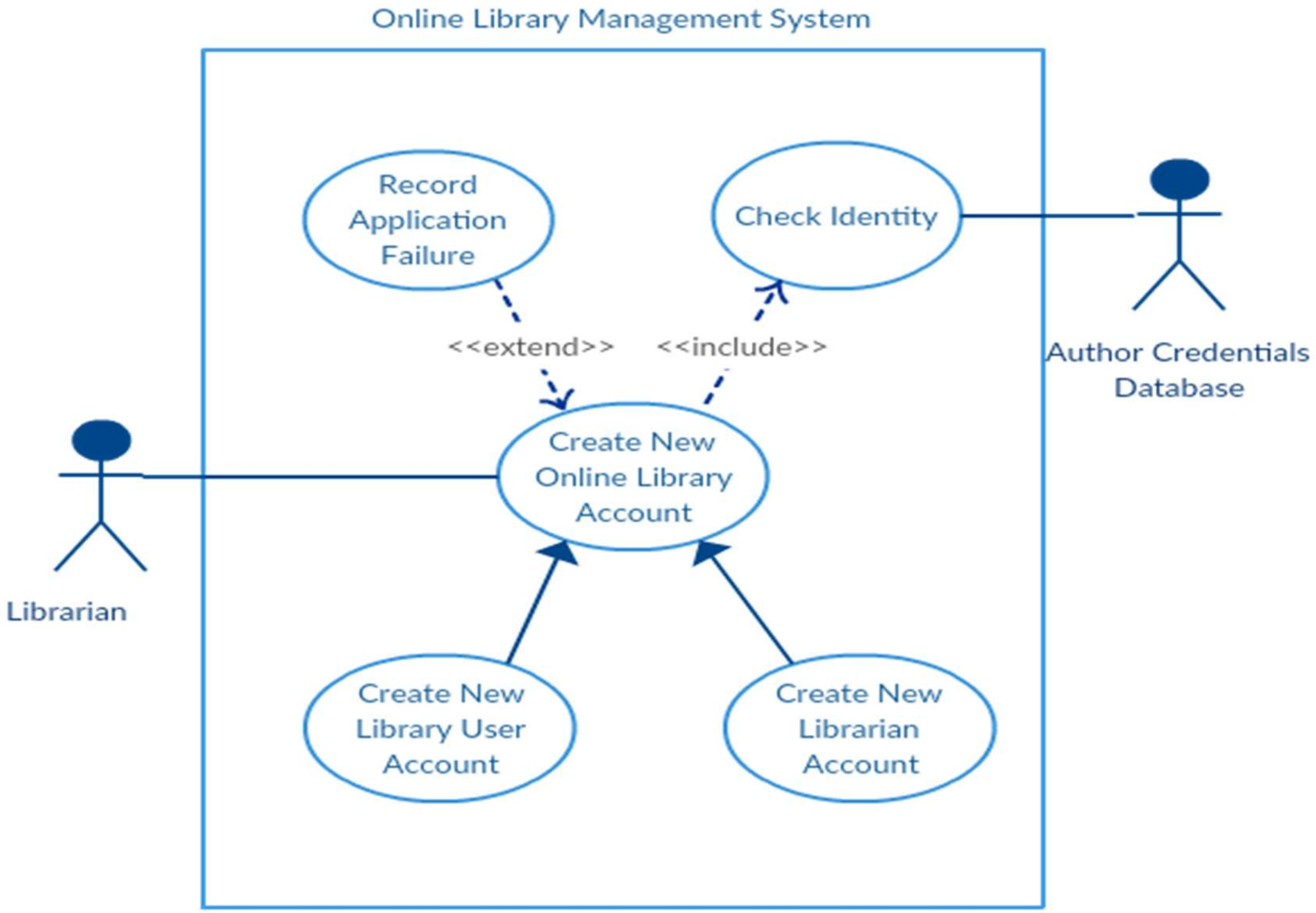

Admin
Student
Teacher
TestPaper
Class
Subject
Division
2: Add/Modify Student()
1: PrepareTestPaper()
6: return
7: return
8: HireTeacher()
5: DisplayClass()
3:DisplaySubjects()
4:DisplayDivision()
9: return
10: AddClass/ModifyClass()
11: return
12: AddSubject/ModifySubject()
13: return
14: AddDivisions/ModifyDivisions()
16: WriteExam()
15: return
17: return
18: MarkAttendance()
19: return value
20: PrepareReportCard()
21: return

Sequence Diagram of an Online Exam System


Admin
Examiner
Student
Test
TestPaper
Result
Response
1: Prepares()
2: return
4: return Status
3: Modify()
5: Gives()
6: return Choice
7: Checks()
8: return
9: SendInvite()
10: return Confirmation
11: SendResult()
12: return Status

How to Draw a Sequence Diagram

- A sequence diagram represents the scenario or flow of events in one single use case.
- The message flow of the sequence diagram is based on the narrative of the particular use case.
- Then, before you start drawing the sequence diagram or decide what interactions should be included in it.
- you need to draw the use case diagram and ready a comprehensive description of what the particular use case does.


Online Library Management System
Record Application Failure
Check Identity
<<extend>>
<<include>>
Author Credentials Database
Create New Online Library Account
Librarian
Create New Library User Account
Create New Librarian Account

- From the above use case diagram example of ‘Create New Online Library Account’, we will focus on the use case named ‘Create New User Account’ to draw our sequence diagram example.
- Before drawing the sequence diagram, it’s necessary to identify the objects or actors that would be involved in creating a new user account.

These would be;

- Librarian
- Online Library Management system
- User credentials database
- Email system
- Once you identify the objects, it is then important to write a detailed description on what the use case does.
- From this description, you can easily figure out the interactions (that should go in the sequence diagram) that would occur between the objects above, once the use case is executed.

- Here are the steps that occur in the use case named ‘Create New Library User Account’.
- The librarian request the system to create a new online library account
- The librarian then selects the library user account type
- The librarian enters the user’s details
- The user’s details are checked using the user Credentials Database

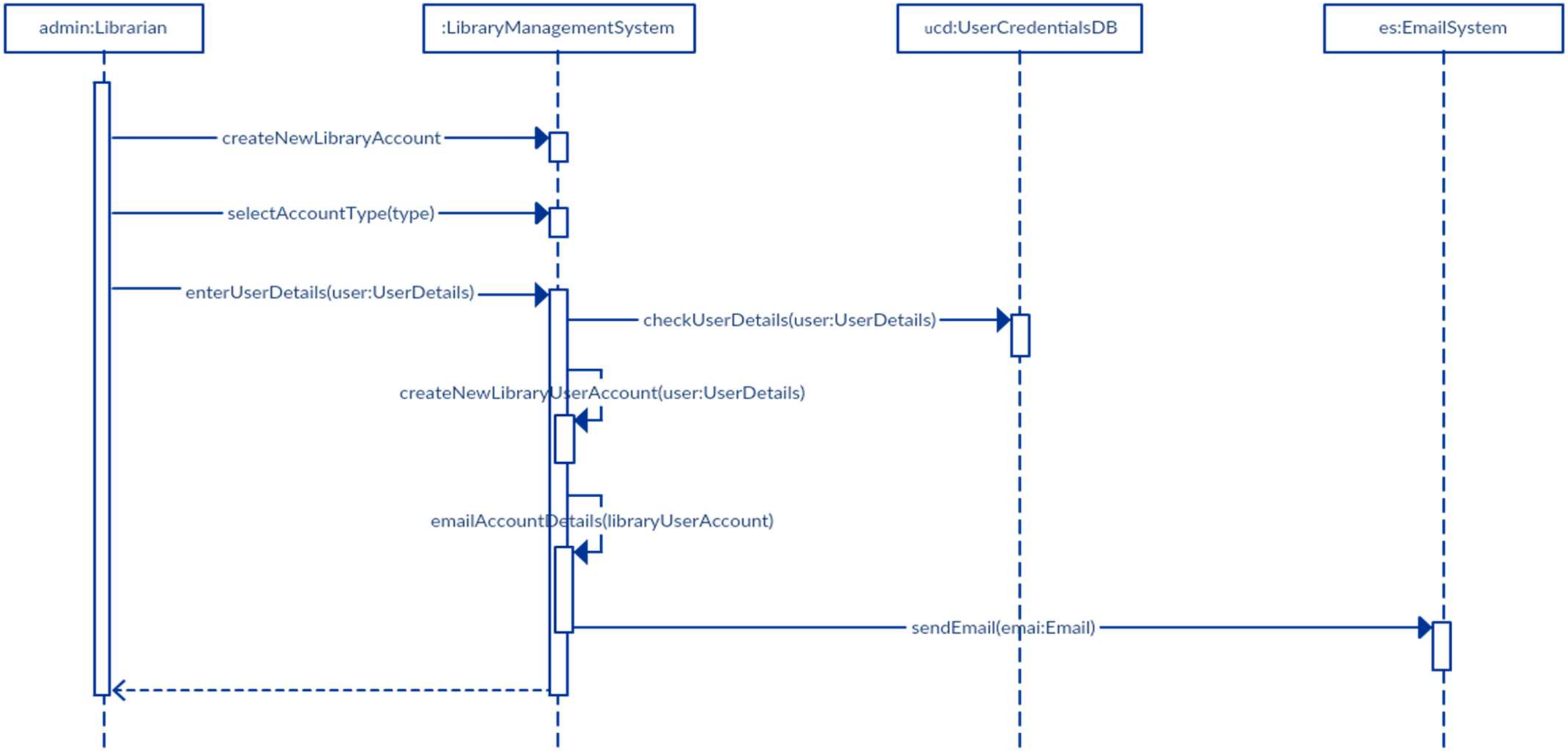
- The new library user account is created

- A summary of the of the new account’s details are then emailed to the user

- From each of these steps, you can easily specify what messages should be exchanged between the objects in the sequence diagram.

- Once it’s clear, you can go ahead and start drawing the sequence diagram.

- The sequence diagram below shows how the objects in the online library management system interact with each other to perform the function ‘Create New Library User Account’.


admin:Librarian
:LibraryManagementSystem
ucd:UserCredentialsDB
es:EmailSystem
createNewLibraryAccount
selectAccountType(type)
enterUserDetails(user:UserDetails)
checkUserDetails(user:UserDetails)
createNewLibraryUserAccount(user:UserDetails)
emailAccountDetails(libraryUserAccount)
sendEmail(emai:Email)

In class exercise

- Draw a sequence diagram for:
    - Draw a sequence diagram for making a peanut butter and jam sandwich if the following objects are alive: knife, peanut butter jar (and peanut butter), jam jar (and jam), bread, plate.
    - We may or may not want the crusts cut off.
    - Don’t forget to open and close things like the jars, and put yourself away, cleanup, etc…

In class exercise

- Draw a sequence diagram for:
  - Returning a movie to Netflix.
  - When you return it a person scan’s it in, the s/w detects that and then goes through a sequence of steps to get the new movie to you.
  - Think of the different classes that would be involved: Queue, Shipping, Inventory, UserAccount, other classes??

In class exercise

- Draw a sequence diagram for:

  - Adding a picture to Flickr (or any online image database).

  - Login, pick an album, upload a picture, etc…

  - Think about the software classes that would be involved – WebGUI
  - (think of this as reporting what the user does), UserAccount, Album, AlbumList, etc…

  - Don’t forget to check and update their current disk usage.

  - For this diagram show the check coming back as acceptable..

  - you would do a second diagram for them running over quota.

In class exercise

- Draw a sequence diagram for:
  - Getting on a flight.
  - Start at home, check in at the counter, go through security, and end up at the gate.
  - (If you have time during the exercise, get yourself to your seat.)
    - You may get searched in security

In class exercise

- Draw a sequence diagram for:
  - Getting money from an ATM machine
    - Treat each part of the ATM as a class
      - Money dispenser
      - Screen
      - Keypad
      - Bank computer
      - Etc…

The Collaboration Diagram

The Collaboration Diagram

- The second interaction diagram is the collaboration diagram.
- It shows the object organization as seen in the following diagram.
- In the collaboration diagram, the method call sequence is indicated by some numbering technique.
- The number indicates how the methods are called one after another.
- We have taken the same order management system to describe the collaboration diagram.

The Collaboration Diagram

- Method calls are similar to that of a sequence diagram.
- However, difference being the sequence diagram does not describe the object organization, whereas the collaboration diagram shows the object organization.
- To choose between these two diagrams, emphasis is placed on the type of requirement.
- If the time sequence is important, then the sequence diagram is used.
- If organization is required, then collaboration diagram is used.

When to use a Collaboration Diagram?

- The collaborations are used when it is essential to depict the relationship between the object.

- Both the sequence and collaboration diagrams represent the same information, but the way of portraying it quite different.

- The collaboration diagrams are best suited for analyzing use cases.

When to use a Collaboration Diagram?

1. To model collaboration among the objects or roles that carry the functionalities of use cases and operations.

2. To model the mechanism inside the architectural design of the system.

3. To capture the interactions that represent the flow of messages between the objects and the roles inside the collaboration.

4. To model different scenarios within the use case or operation, involving a collaboration of several objects and interactions.

When to use a Collaboration Diagram?

5. To support the identification of objects participating in the use case.

6. In the collaboration diagram, each message constitutes a sequence number, such that the top-level message is marked as one and so on.

7. The messages sent during the same call are denoted with the same decimal prefix, but with different suffixes of 1, 2, etc. as per their occurrence.

Notations of a Collaboration Diagram

- Following are the components of a component diagram that are enlisted below:

1. Objects: The representation of an object is done by an object symbol with its name and class underlined, separated by a colon.

In the collaboration diagram, objects are utilized in the following ways:

1. The object is represented by specifying their name and class.
2. It is not mandatory for every class to appear.
3. A class may constitute more than one object.
4. In the collaboration diagram, firstly, the object is created, and then its class is specified.
5. To differentiate one object from another object, it is necessary to name them.

Notations of a Collaboration Diagram

2. Actors: In the collaboration diagram, the actor plays the main role as it invokes the interaction.

- Each actor has its respective role and name.
- In this, one actor initiates the use case.

3. Links: The link is an instance of association, which associates the objects and actors.

- It portrays a relationship between the objects through which the messages are sent.
- It is represented by a solid line.
- The link helps an object to connect with or navigate to another object, such that the message flows are attached to links.

Notations of a Collaboration Diagram

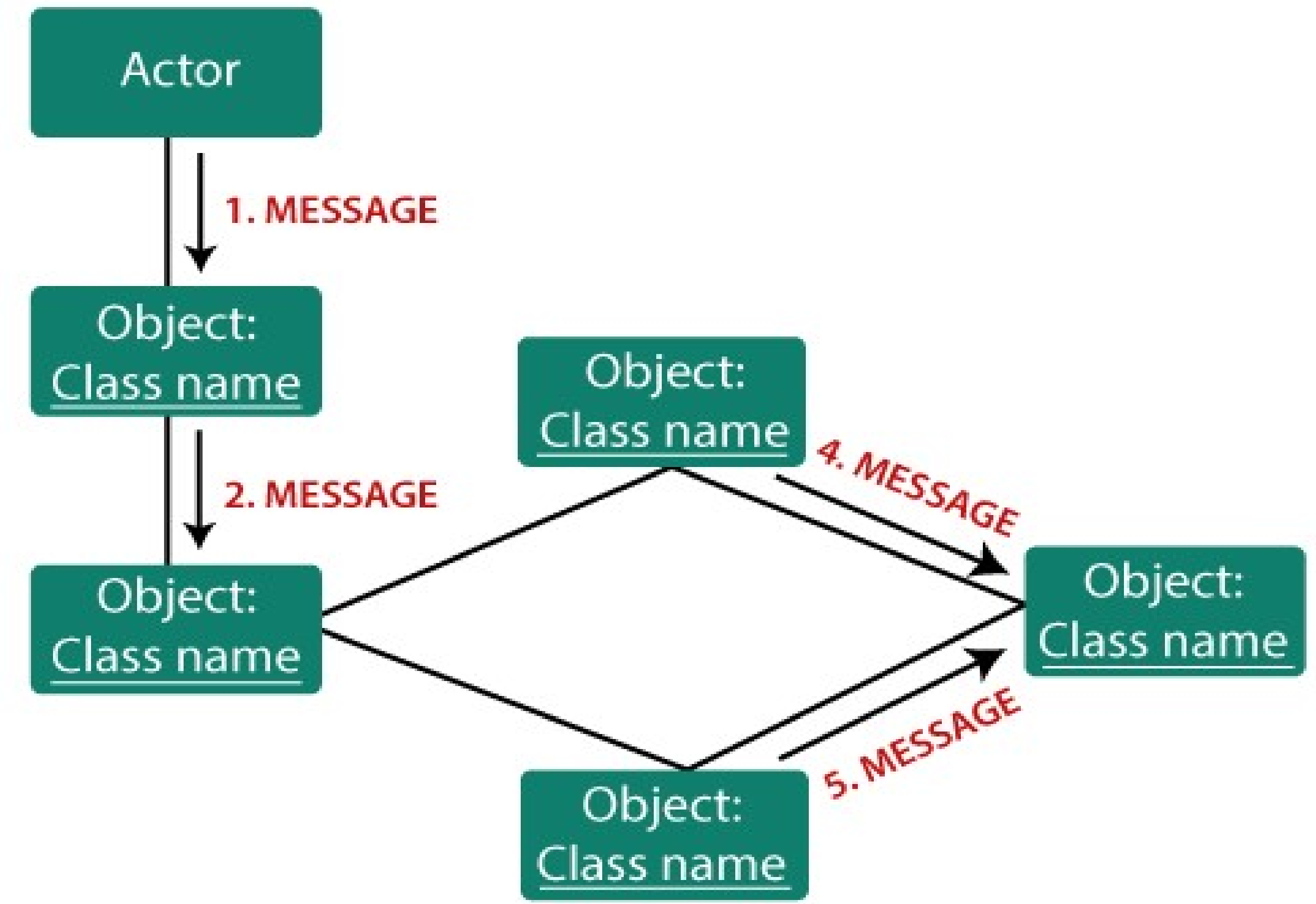
4. Messages: It is a communication between objects which carries information and includes a sequence number, so that the activity may take place.

- It is represented by a labeled arrow, which is placed near a link.

- The messages are sent from the sender to the receiver, and the direction must be navigable in that particular direction.

- The receiver must understand the message.

Components of a collaboration diagram


Actor
1. MESSAGE
Object:
Class name
2. MESSAGE
Object:
Class name
Object:
Class name
4. MESSAGE
Object:
Class name
5. MESSAGE
Object:
Class name

Steps for creating a Collaboration Diagram

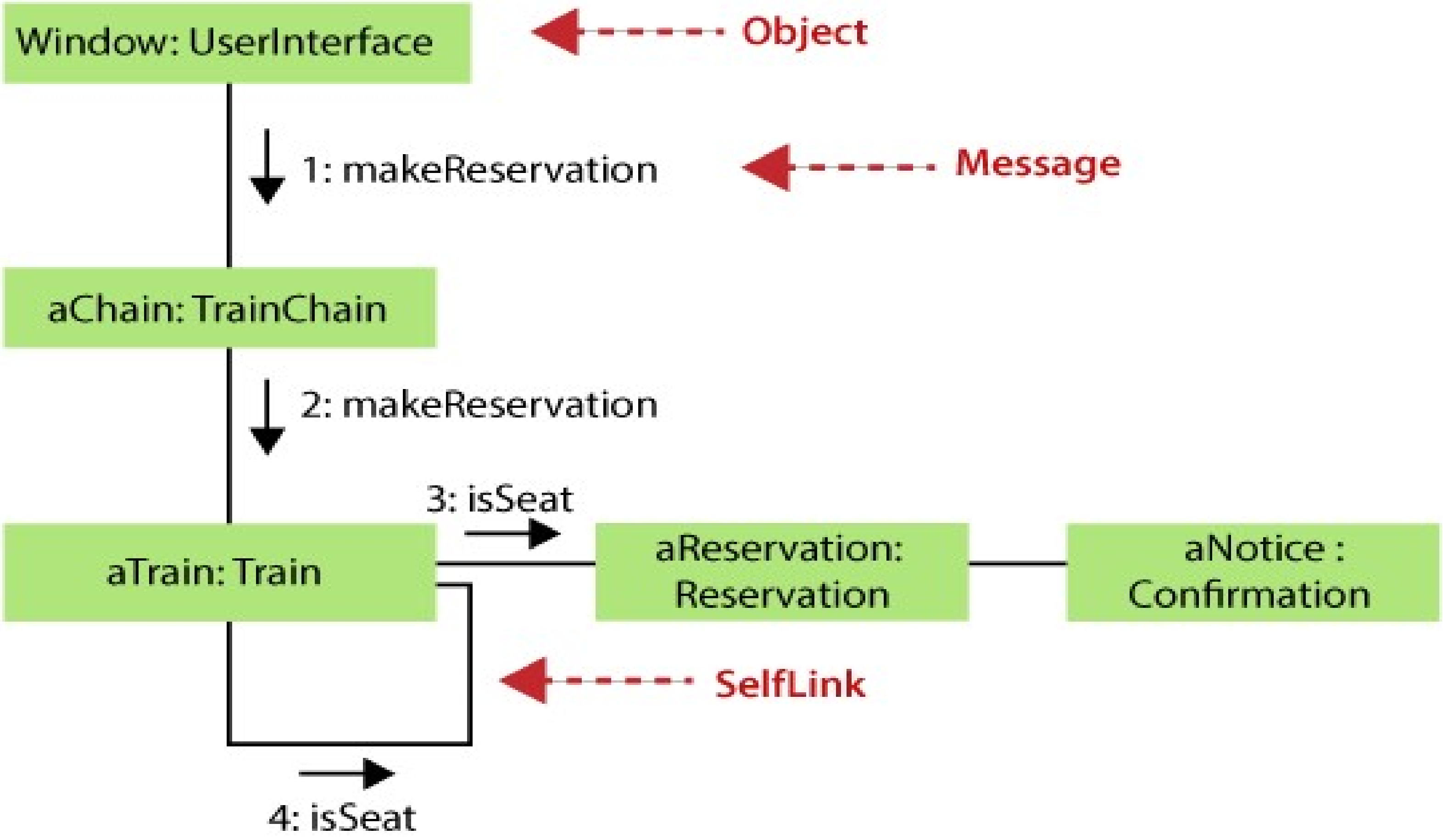
1. Determine the behavior for which the realization and implementation are specified.

2. Discover the structural elements that are class roles, objects, and subsystems for performing the functionality of collaboration.

    1. Choose the context of an interaction: system, subsystem, use case, and operation.

3. Think through alternative situations that may be involved.
    1. Implementation of a collaboration diagram at an instance level, if needed.
    2. A specification level diagram may be made in the instance level sequence diagram for summarizing alternative situations.

Example


Window: UserInterface
Object
1: makeReservation
Message
aChain: TrainChain
2: makeReservation
3: isSeat
aTrain: Train
aReservation: Reservation
aNotice : Confirmation
SelfLink
4: isSeat

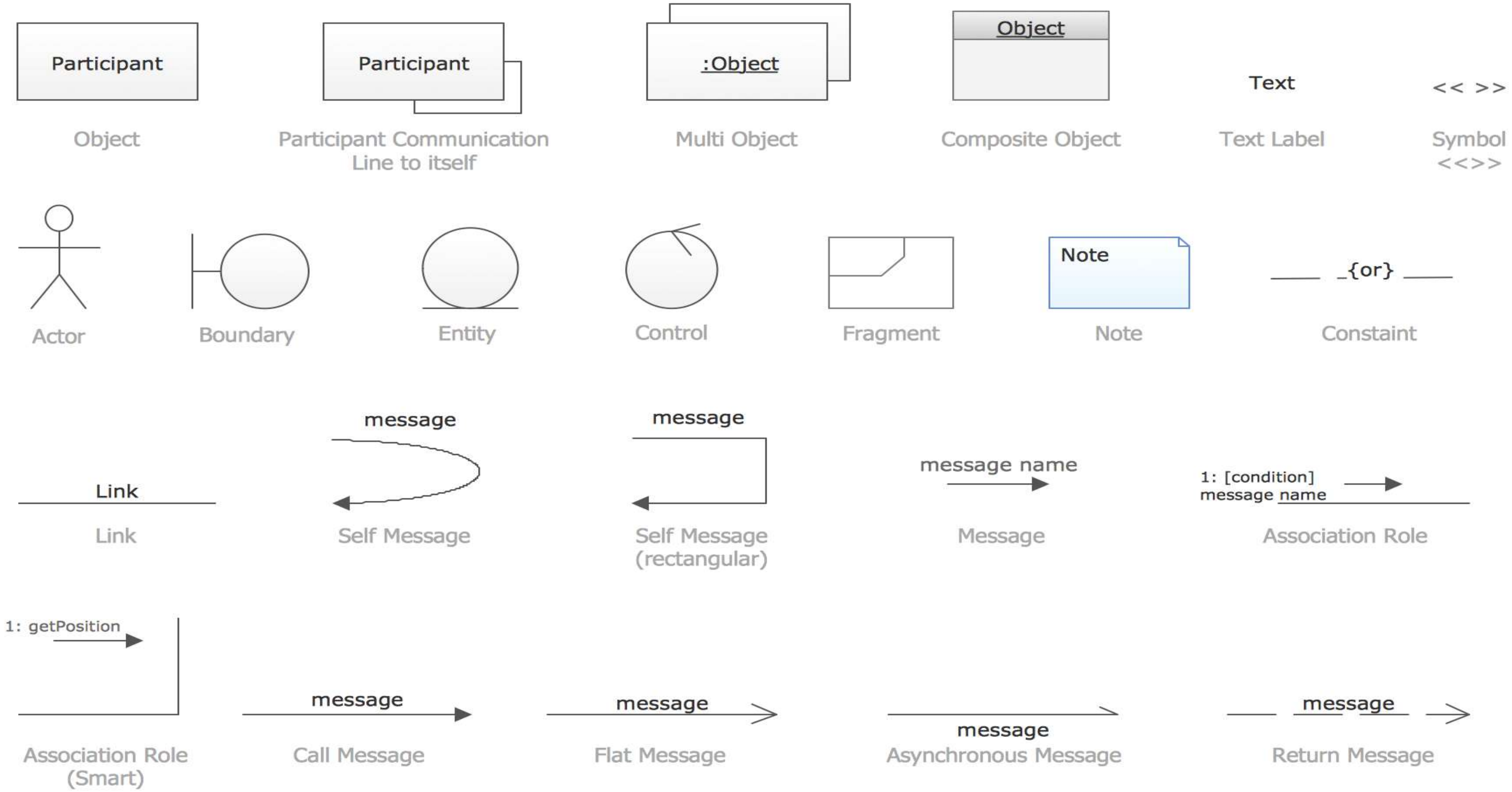
Collaboration Design Elements

- Object
- Lifeline (actor)
- Lifeline (boundary)
- Lifeline (entity)
- Lifeline (control)
- Participant communication line to itself
- Link
- Message
- Association role
- Fragment
- Symbol << >>
- Note
- Association role (Smart)

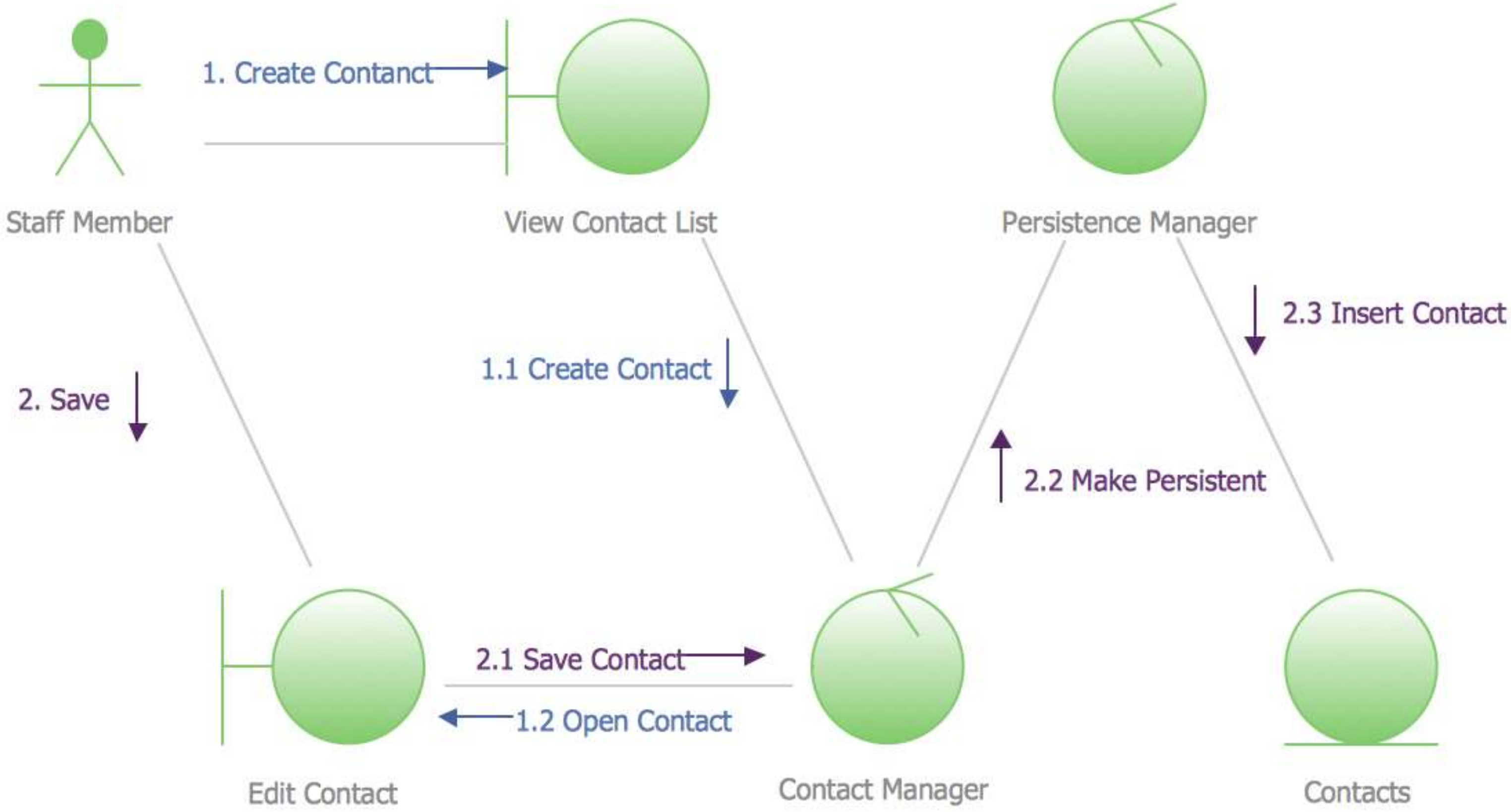
Collaboration Design Elements

- Multi object
- Composite object
- Return message
- Call message
- Asynchronous message
- Flat message
- Self message
- Self message (rectangular)
- Constraint
- Text label

UML Collaboration Diagram. Design Elements

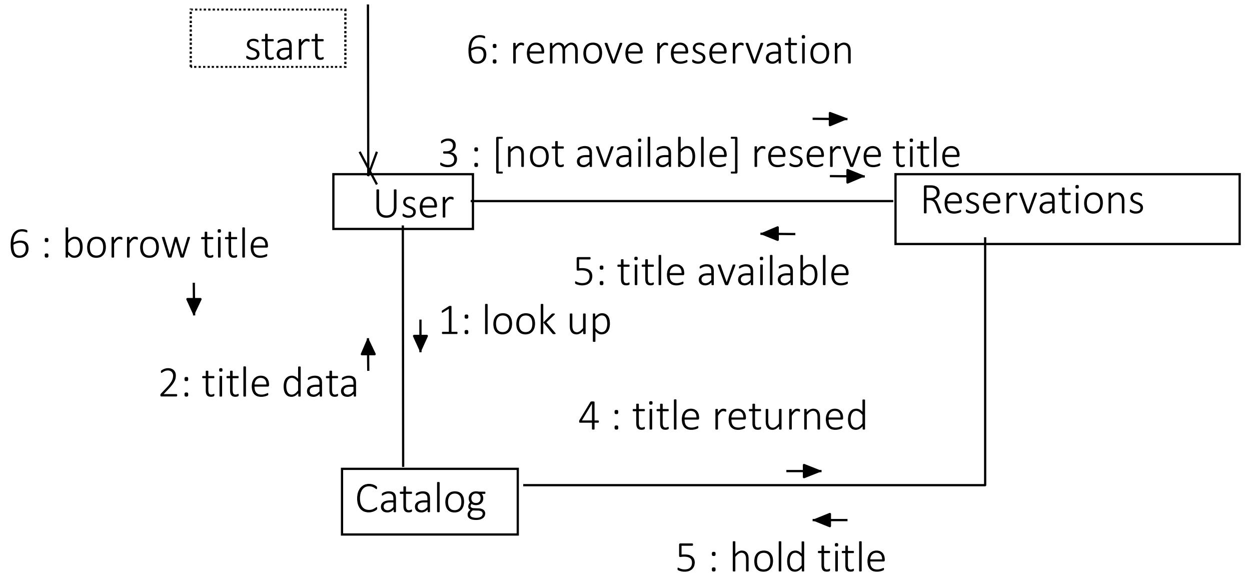

Participant
Object
Participant
Participant Communication Line to itself
:Object
Multi Object
Object
Composite Object
Text
Text Label
<< >>
Symbol <<>>
Actor
Boundary
Entity
Control
Fragment
Note
Note
_{or}
Constaint
Link
Link
message
Self Message
message
Self Message (rectangular)
message name
Message
1: [condition] message name
Association Role
1: getPosition
Association Role (Smart)
message
Call Message
message
Flat Message
message
Asynchronous Message
message
Return Message

Example


Staff Member
1. Create Contanct
View Contact List
Persistence Manager
2.3 Insert Contact
1.1 Create Contact
2. Save
2.2 Make Persistent
2.1 Save Contact
1.2 Open Contact
Edit Contact
Contact Manager
Contacts

The creation process of the contact list and can be used at the staff training and staff working, at the attraction process the new clients.

Interaction Diagrams: Collaboration diagrams


start
6: remove reservation
3 : [not available] reserve title
User
Reservations
6 : borrow title
5: title available
1: look up
2: title data
4 : title returned
Catalog
5 : hold title

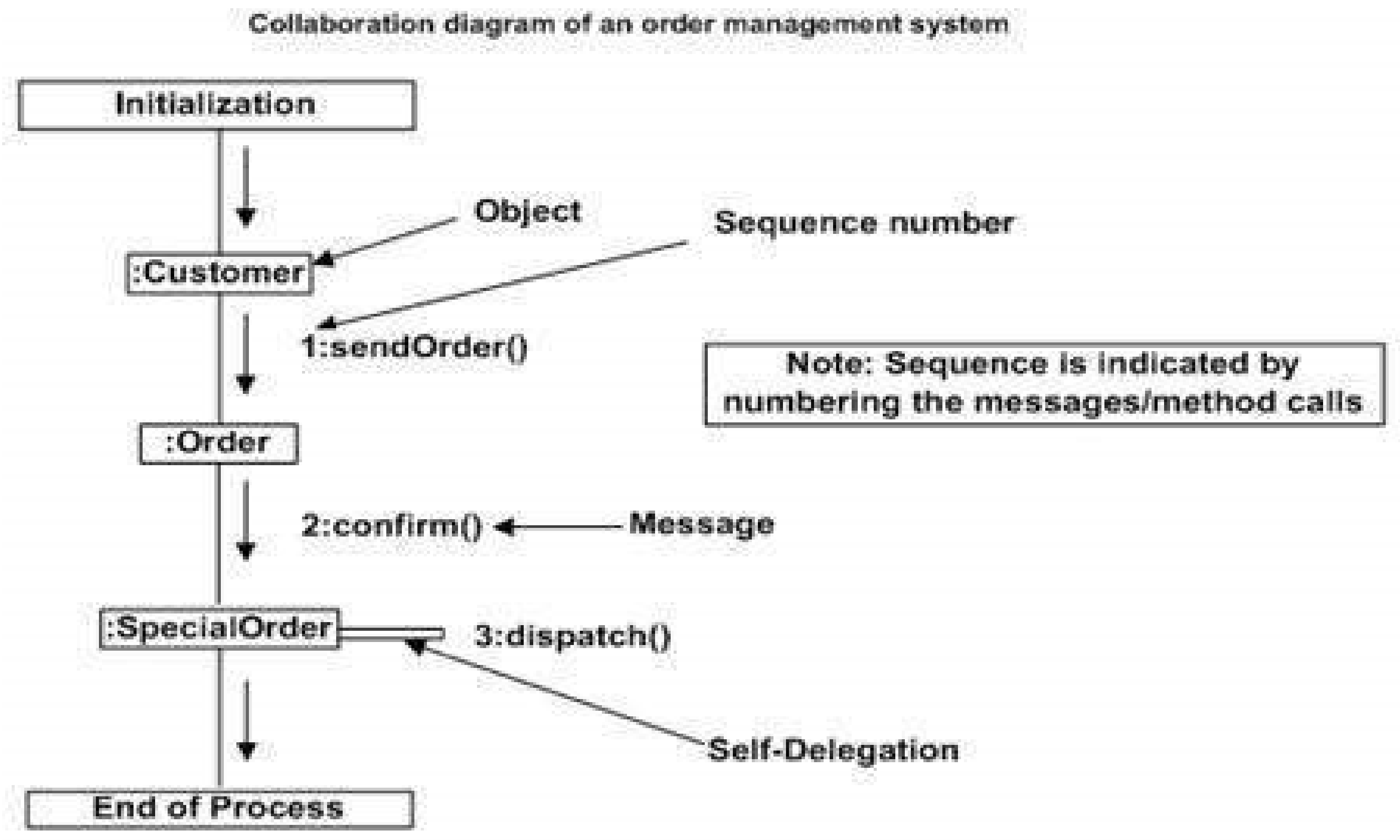
Collaboration diagrams

- Collaboration diagrams are equivalent to sequence diagrams.

- All the features of sequence diagrams are equally applicable to collaboration diagrams

- Use a sequence diagram when the transfer of information is the focus of attention

- Use a collaboration diagram when concentrating on the classes


Collaboration diagram of an order management system

Initialization
Object
Sequence number
:Customer
1:sendOrder()
:Order
2:confirm()
Message
:SpecialOrder
3:dispatch()
Self-Delegation
End of Process

Note: Sequence is indicated by numbering the messages/method calls

Benefits of a Collaboration Diagram

- The collaboration diagram is also known as Communication Diagram.

- It mainly puts emphasis on the structural aspect of an interaction diagram, i.e., how lifelines are connected.

- The syntax of a collaboration diagram is similar to the sequence diagram; just the difference is that the lifeline does not consist of tails.

- The messages transmitted over sequencing is represented by numbering each individual message.

- The collaboration diagram is semantically weak in comparison to the sequence diagram.

Benefits of a Collaboration Diagram

- The special case of a collaboration diagram is the object diagram.

- It focuses on the elements and not the message flow, like sequence diagrams.

- Since the collaboration diagrams are not that expensive, the sequence diagram can be directly converted to the collaboration diagram.

- There may be a chance of losing some amount of information while implementing a collaboration diagram with respect to the sequence diagram.

Collaboration Diagram- Limitations

1. Multiple objects residing in the system can make a complex collaboration diagram, as it becomes quite hard to explore the objects.

2. It is a time-consuming diagram.

3. After the program terminates, the object is destroyed.

4. As the object state changes momentarily, it becomes difficult to keep an eye on every single state that has occurred inside the object of a system.

BITS Pilani
Pilani Campus

SOFTWARE DEVELOPMENT LIFECYCLE

A R Rahman

Unified Modeling Language for Dynamic Analysis

Unified Modeling Language (UML)
Activity Diagrams

Activity

World Artistic & Freestyle

Activity Diagrams

- We use Activity Diagrams to illustrate the flow of control in a system and refer to the steps involved in the execution of a use case.
- We model sequential and concurrent activities using activity diagrams.
- So, we basically depict workflows visually using an activity diagram.
- An activity diagram focuses on condition of flow and the sequence in which it happens.
- We describe or depict what causes a particular event using an activity diagram.
- UML models basically three types of diagrams, namely, structure diagrams, interaction diagrams, and behavior diagrams.
- An activity diagram is a behavioral diagram i.e. it depicts the behavior of a system.

Activity Diagrams

- UML models basically three types of diagrams, namely, structure diagrams, interaction diagrams, and behavior diagrams.
- An activity diagram is a behavioral diagram i.e. it depicts the behavior of a system.
- An activity diagram portrays the control flow from a start point to a finish point showing the various decision paths that exist while the activity is being executed.
- We can depict both sequential processing and concurrent processing of activities using an activity diagram.
- They are used in business and process modelling where their primary use is to depict the dynamic aspects of a system.

Difference between an Activity diagram and a Flowchart

- An activity diagram is very similar to a flowchart.
- So let us understand if an activity diagrams or a flowcharts are any different :
- Flowcharts were typically invented earlier than activity diagrams.
- Non programmers use Flow charts to model workflows.
- For example: A manufacturer uses a flow chart to explain and illustrate how a particular product is manufactured.

Difference between an Activity diagram and a Flowchart

- We can call a flowchart a primitive version of an activity diagram.
- Business processes where decision making is involved is expressed using a flow chart.
- So, programmers use activity diagrams (advanced version of a flowchart) to depict workflows.
- An activity diagram is used by developers to understand the flow of programs on a high level.
- It also enables them to figure out constraints and conditions that cause particular events.
- A flow chart converges into being an activity diagram if complex decisions are being made.

Difference between an Activity diagram and a Flowchart

- We need to convey a lot of information with clarity and make sure it is short.
- So an activity diagram helps people on both sides i.e. Businessmen and Developers to interact and understand systems.

Do we need to use both the diagram and the textual documentation?

- Different individuals have different preferences in which they understand something.

- For example: To understand a concept, some people might prefer a written tutorial with images while others would prefer a video lecture.
-
  So we generally use both the diagram and the textual documentation to make our system description as clear as possible.

- We also need to be sensitive to the needs of the audience that we are catering to at times.

Difference between a Use case diagram and an Activity diagram

- An activity diagram is used to model the workflow depicting conditions, constraints, sequential and concurrent activities.

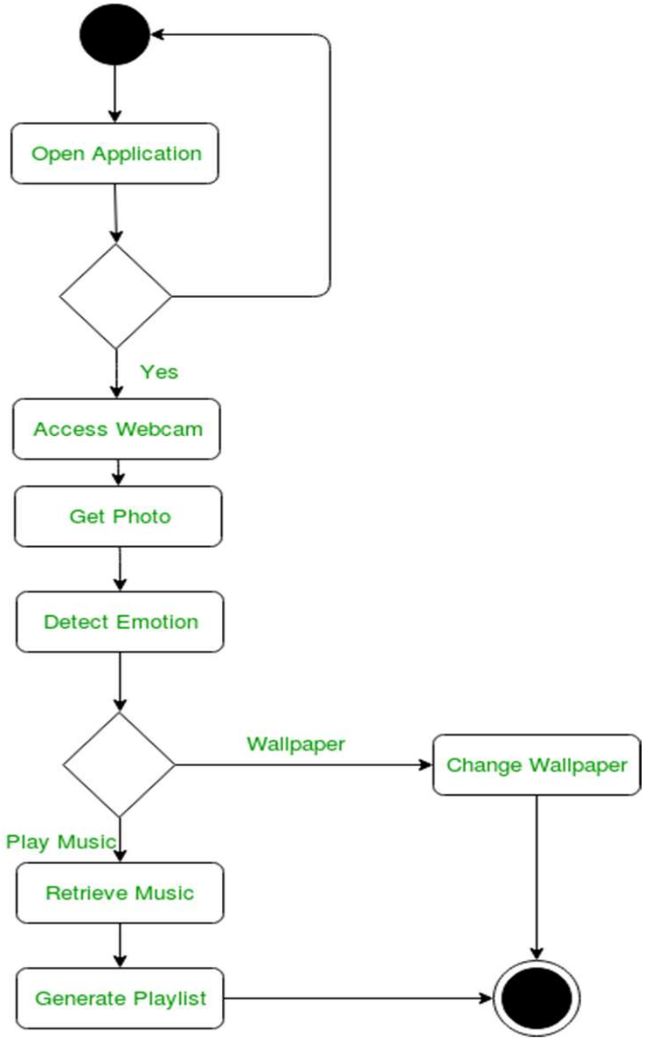
- On the other hand, the purpose of a Use Case is to just depict the functionality i.e. what the system does and not how it is done.

- So in simple terms, an activity diagram shows ‘How’ while a Use case shows ‘What’ for a particular system.

- The levels of abstraction also vary for both of them.

- An activity diagram can be used to illustrate a business process (high level implementation) to a stand alone algorithm (ground level implementation).

- However, Use cases have a low level of abstraction. They are used to show a high level of implementation only.


Open Application
Yes
Access Webcam
Get Photo
Detect Emotion
Wallpaper
Change Wallpaper
Play Music
Retrieve Music
Generate Playlist

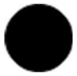
an activity diagram for an emotion based music player

This activity diagram for an emotion based music player which can also be used to change the wallpaper.

components used in the diagram and the standard notations

- Activity Diagram Notations –

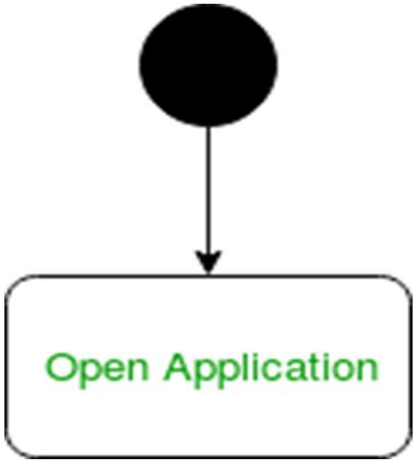
1. **Initial State –** The starting state before an activity takes place is depicted using the initial state.



- notation for initial state or start state
- A process can have only one initial state unless we are depicting nested activities.
- We use a black filled circle to depict the initial state of a system.
- For objects, this is the state when they are instantiated.
- The Initial State from the UML Activity Diagram marks the entry point and the initial Activity State.

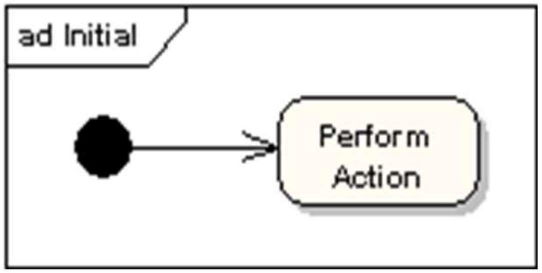
For example

- Here the initial state is the state of the system before the application is opened.



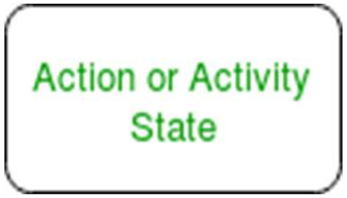
initial state symbol being used

An initial or start node is depicted by a large black spot, as shown below.


ad Initial
Perform Action

Action or Activity State

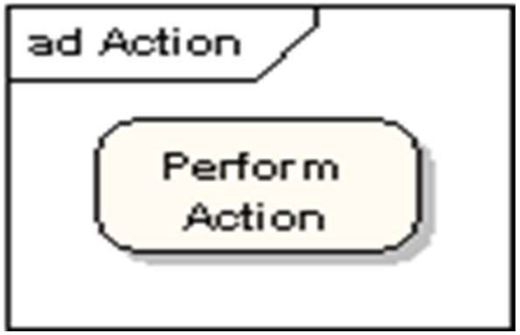
- An activity represents execution of an action on objects or by objects.
- We represent an activity using a rectangle with rounded corners.
- Basically any action or event that takes place is represented using an activity.



notation for an activity state

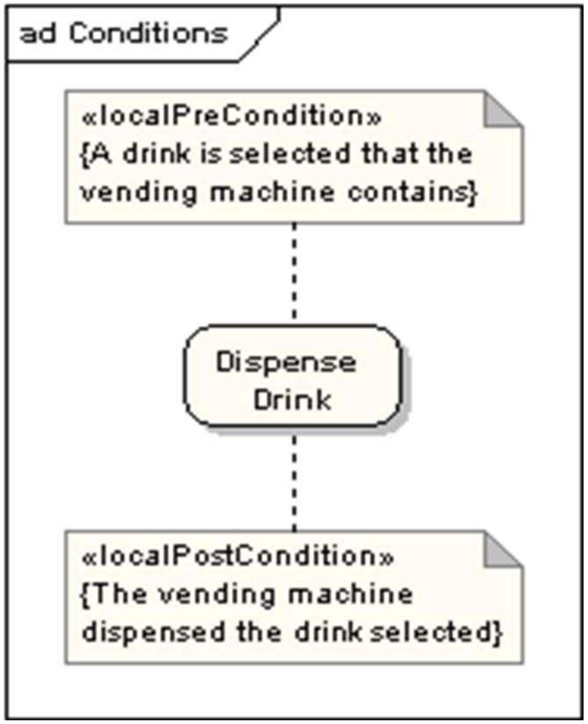
Actions

- An action represents a single step within an activity. Actions are denoted by round-cornered rectangles.


ad Action
Perform Action

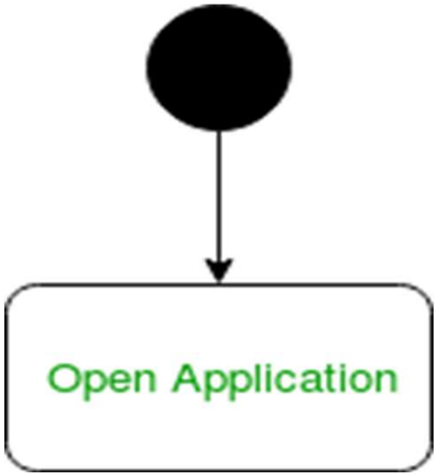
Action Constraints

- Constraints can be attached to an action.
- The following diagram shows an action with local pre- and post-conditions.


ad Conditions
«localPreCondition»
{A drink is selected that the vending machine contains}
Dispense Drink
«localPostCondition»
{The vending machine dispensed the drink selected}

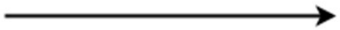
- Consider the previous example of opening an application opening the application is an activity state in the activity diagram.

- activity state symbol being used



Action Flow or Control flows

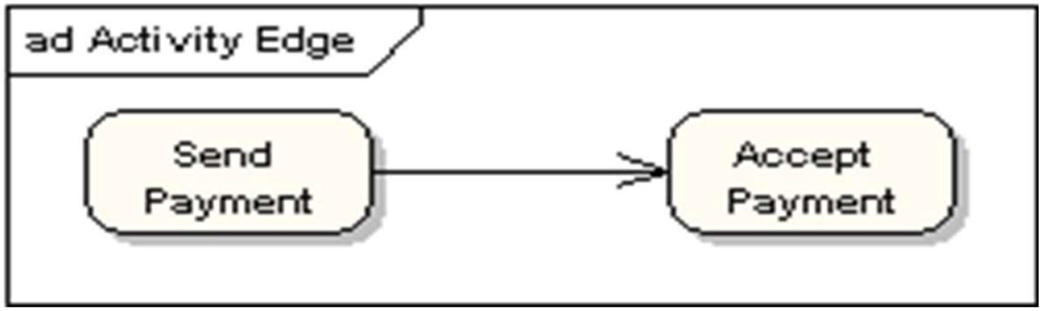
- Action flows or Control flows are also referred to as paths and edges.
- They are used to show the transition from one activity state to another.
- notation for control Flow



- An activity state can have multiple incoming and outgoing action flows.
- We use a line with an arrow head to depict a Control Flow.
- If there is a constraint to be adhered to while making the transition it is mentioned on the arrow.

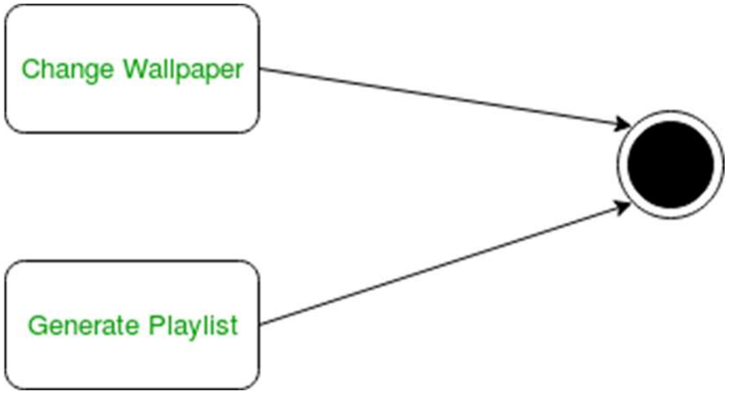
Control Flow

- A control flow shows the flow of control from one action to the next. Its notation is a line with an arrowhead.


ad Activity Edge
Send Payment
Accept Payment

Example

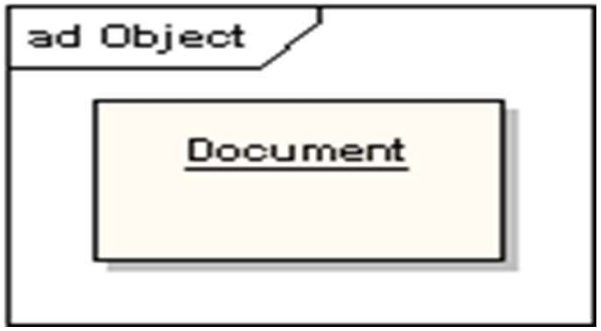
- Here both the states transit into one final state using action flow symbols i.e. arrows.


Change Wallpaper
Generate Playlist

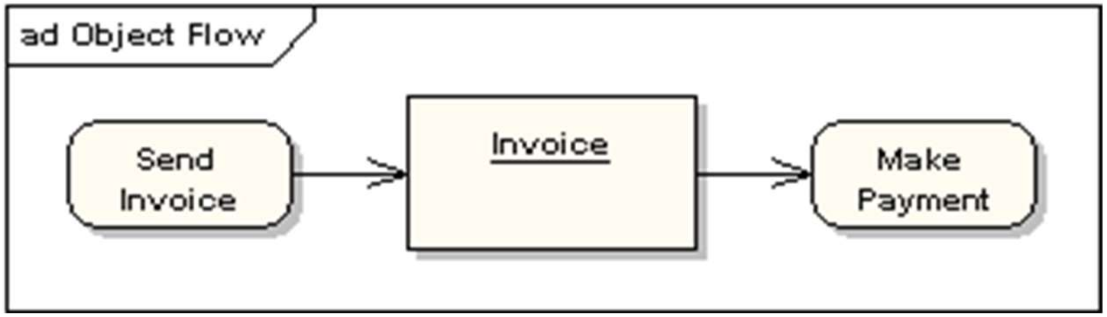
using action flows for transitions

Objects and Object Flows

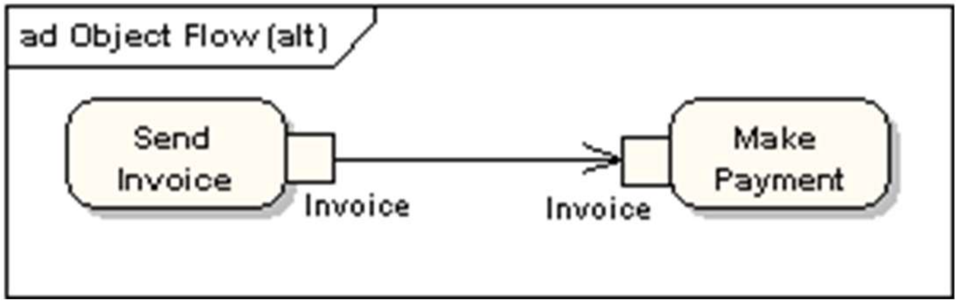
- An object flow is a path along which objects or data can pass. An object is shown as a rectangle.



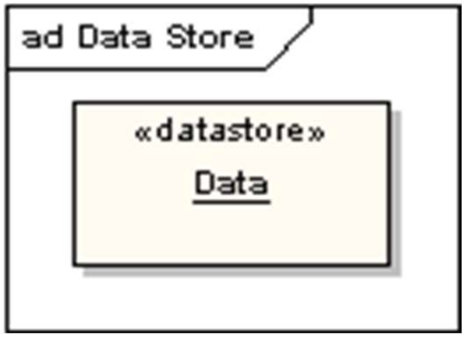
- An object flow is shown as a connector with an arrowhead denoting the direction the object is being passed.


ad Object Flow
Send Invoice
Invoice
Make Payment

- An object flow must have an object on at least one of its ends.
- A shorthand notation for the above diagram would be to use input and output pins.

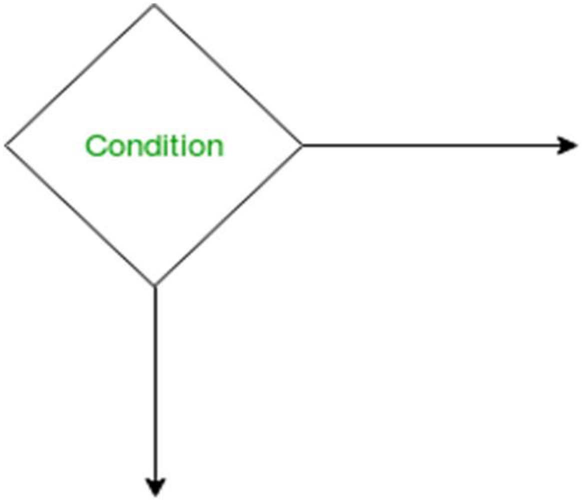

ad Object Flow (alt)
Send Invoice
Invoice
Invoice
Make Payment

- A data store is shown as an object with the «datastore» keyword.


ad Data Store
«datastore»
Data

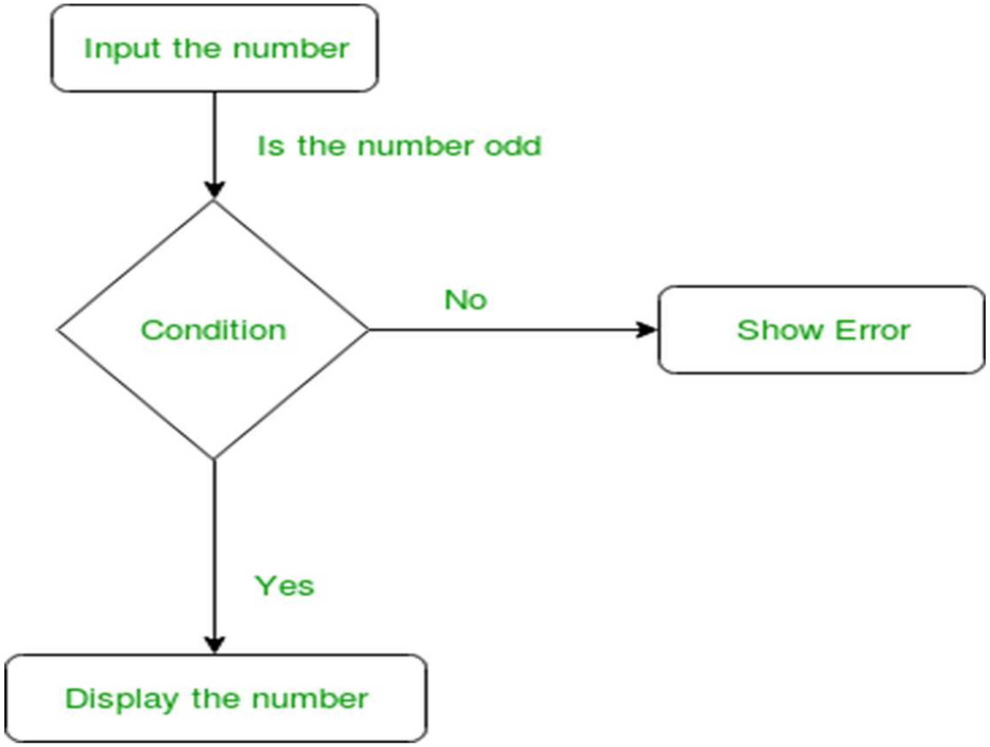
Decision node and Branching

- When we need to make a decision before deciding the flow of control, we use the decision node.
- notation for decision node



- The outgoing arrows from the decision node can be labelled with conditions or guard expressions.
- It always includes two or more output arrows.

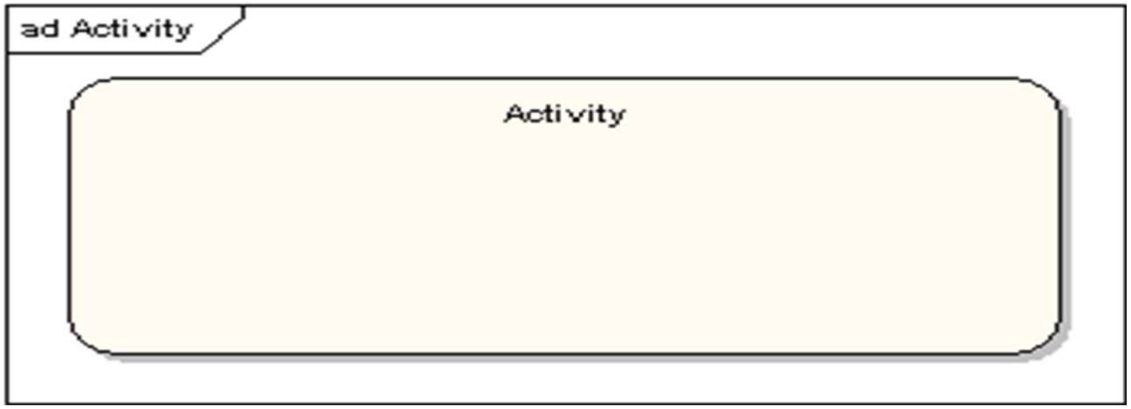
An activity diagram using decision node


Input the number
Is the number odd
Condition
No
Show Error
Yes
Display the number

Activities

- An activity is the specification of a parameterized sequence of behaviour.

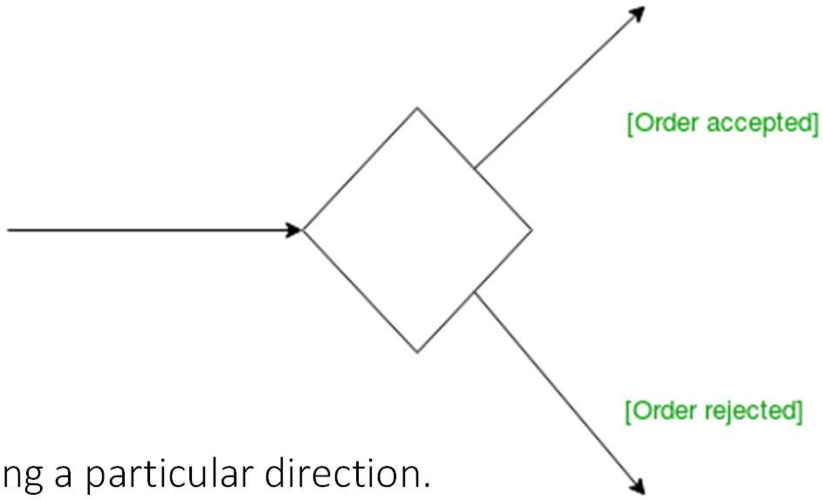
- An activity is shown as a round-cornered rectangle enclosing all the actions, control flows and other elements that make up the activity.



Guard condition a.k.a Guards

- **Guards –** A Guard refers to a statement written next to a decision node on an arrow sometimes within square brackets.

- guards being used next to a decision node



- The statement must be true for the control to shift along a particular direction.

- Guards help us know the constraints and conditions which determine the flow of a process.

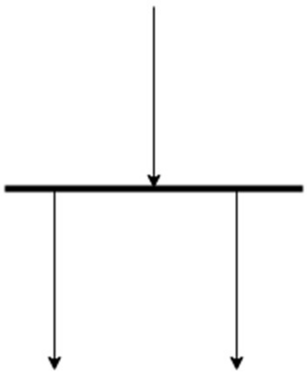
Fork

- Fork nodes are used to support concurrent activities.

- When we use a fork node when both the activities get executed concurrently i.e. no decision is made before splitting the activity into two parts.

- Both parts need to be executed in case of a fork statement.

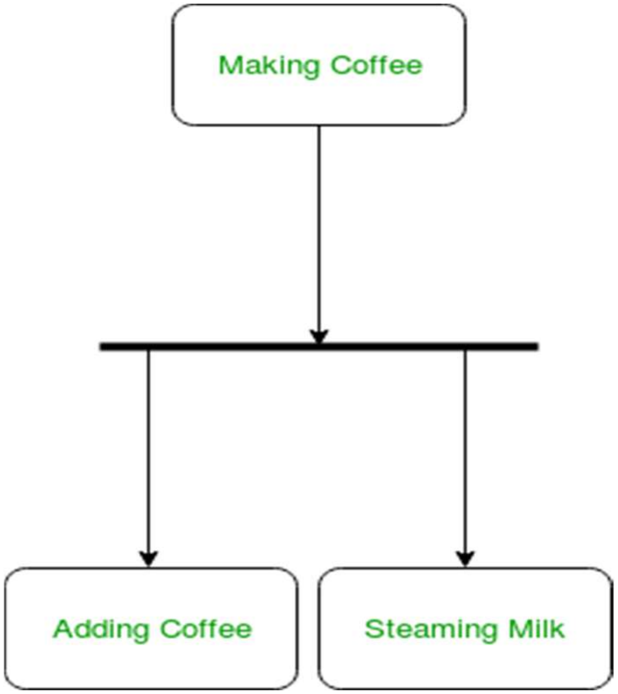
- We use a rounded solid rectangular bar to represent a Fork notation with incoming arrow from the parent activity state and outgoing arrows towards the newly created activities.



fork notation

For example:

- The activity of making coffee can be split into two concurrent activities and hence we use the fork notation.

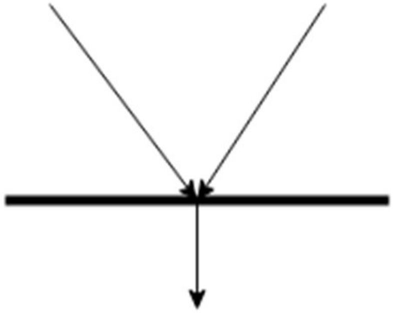

Making Coffee
Adding Coffee
Steaming Milk

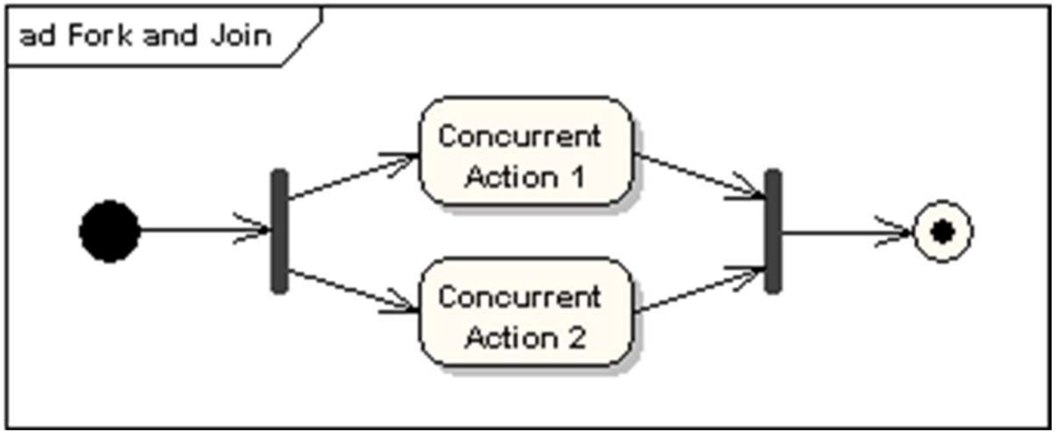
a diagram using fork

Joins

- Join nodes are used to support concurrent activities converging into one.
- For join notations we have two or more incoming edges and one outgoing edge.

join notation




ad Fork and Join
Concurrent Action 1
Concurrent Action 2

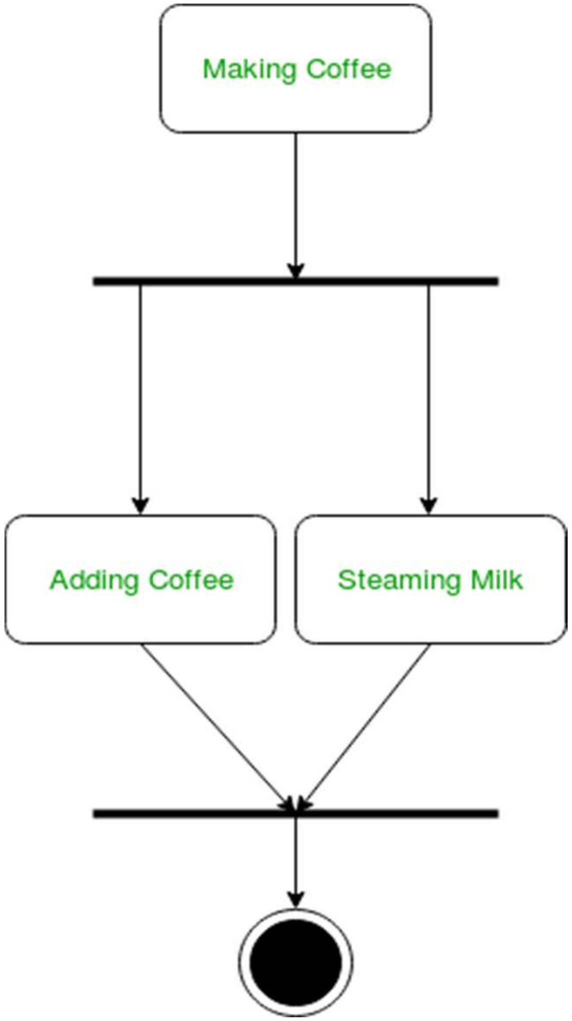
- Forks and joins have the same notation: either a horizontal or vertical bar (the orientation is dependent on whether the control flow is running left to right or top to bottom).

- They indicate the start and end of concurrent threads of control.

For example

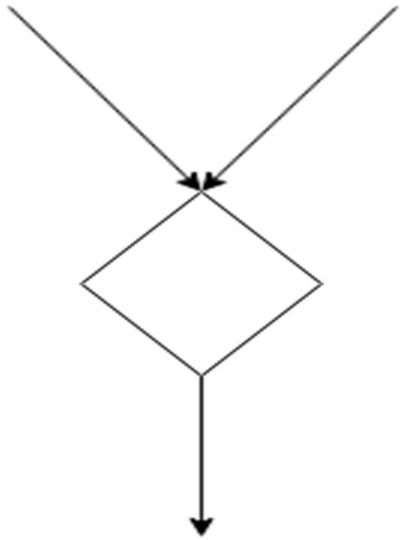
When both activities i.e. steaming the milk and adding coffee get completed, we converge them into one final activity.

a diagram using join notation


Making Coffee
Adding Coffee
Steaming Milk

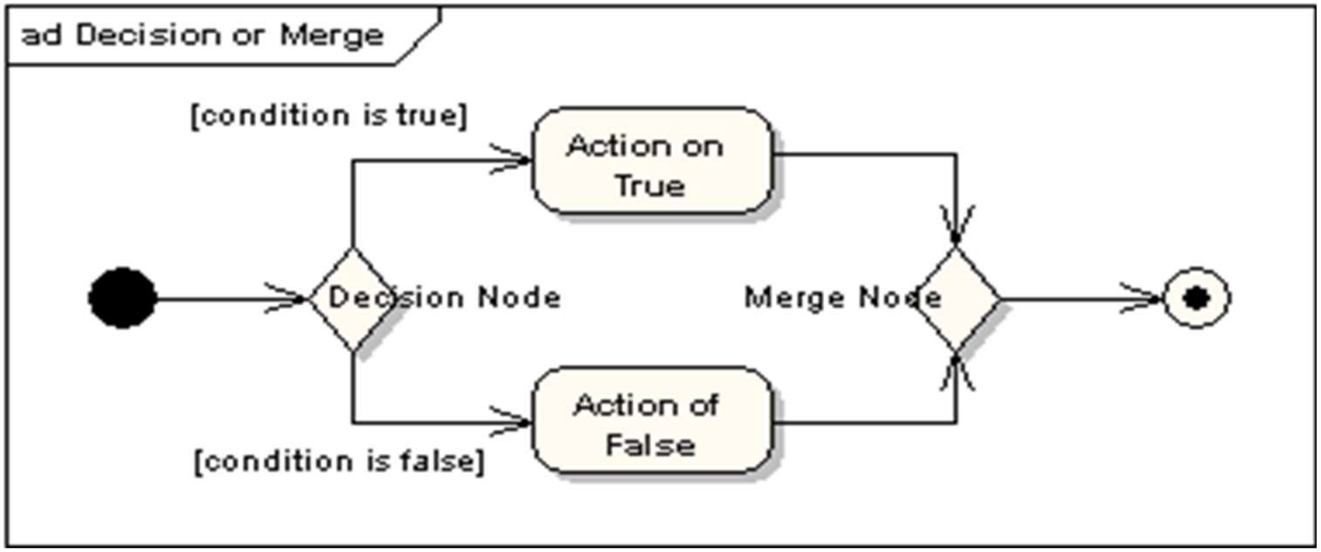
Merge or Merge Event

- Scenarios arise when activities which are not being executed concurrently have to be merged.
- We use the merge notation for such scenarios.
- merge notation



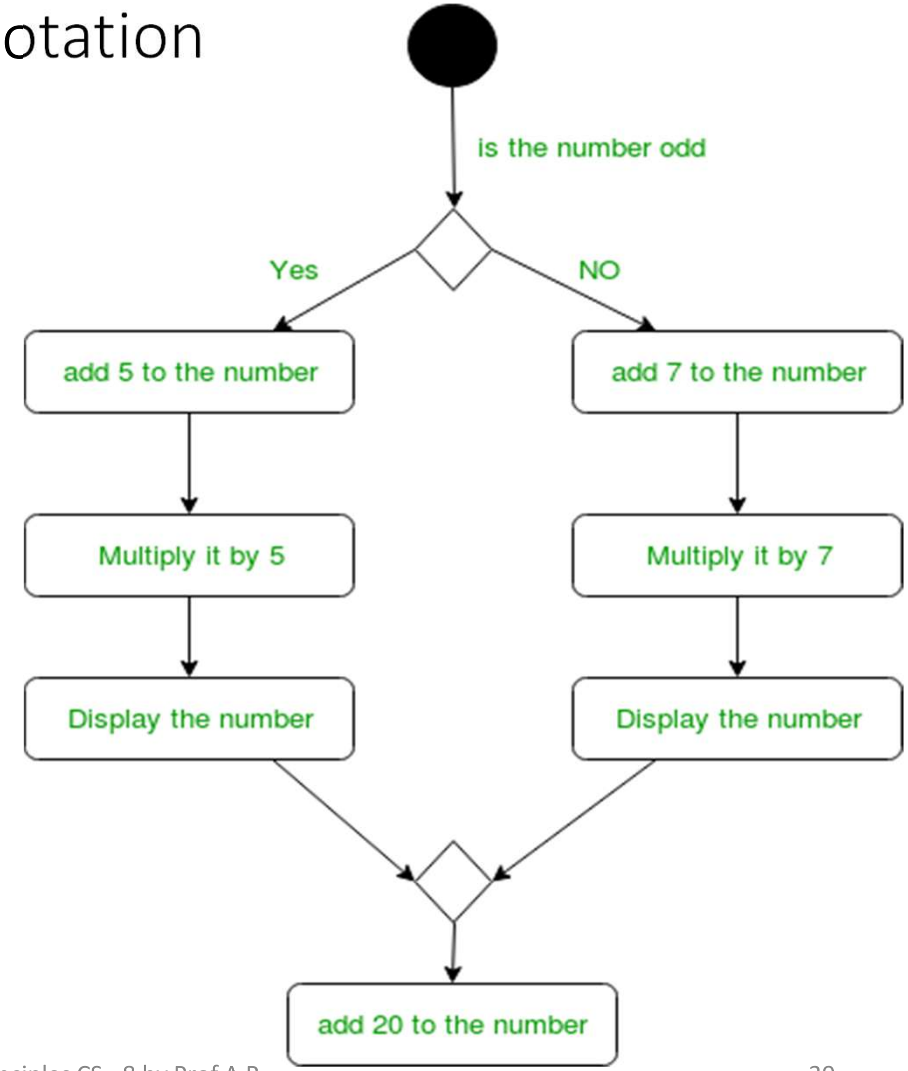
- We can merge two or more activities into one if the control proceeds onto the next activity irrespective of the path chosen.

An activity diagram using merge notation

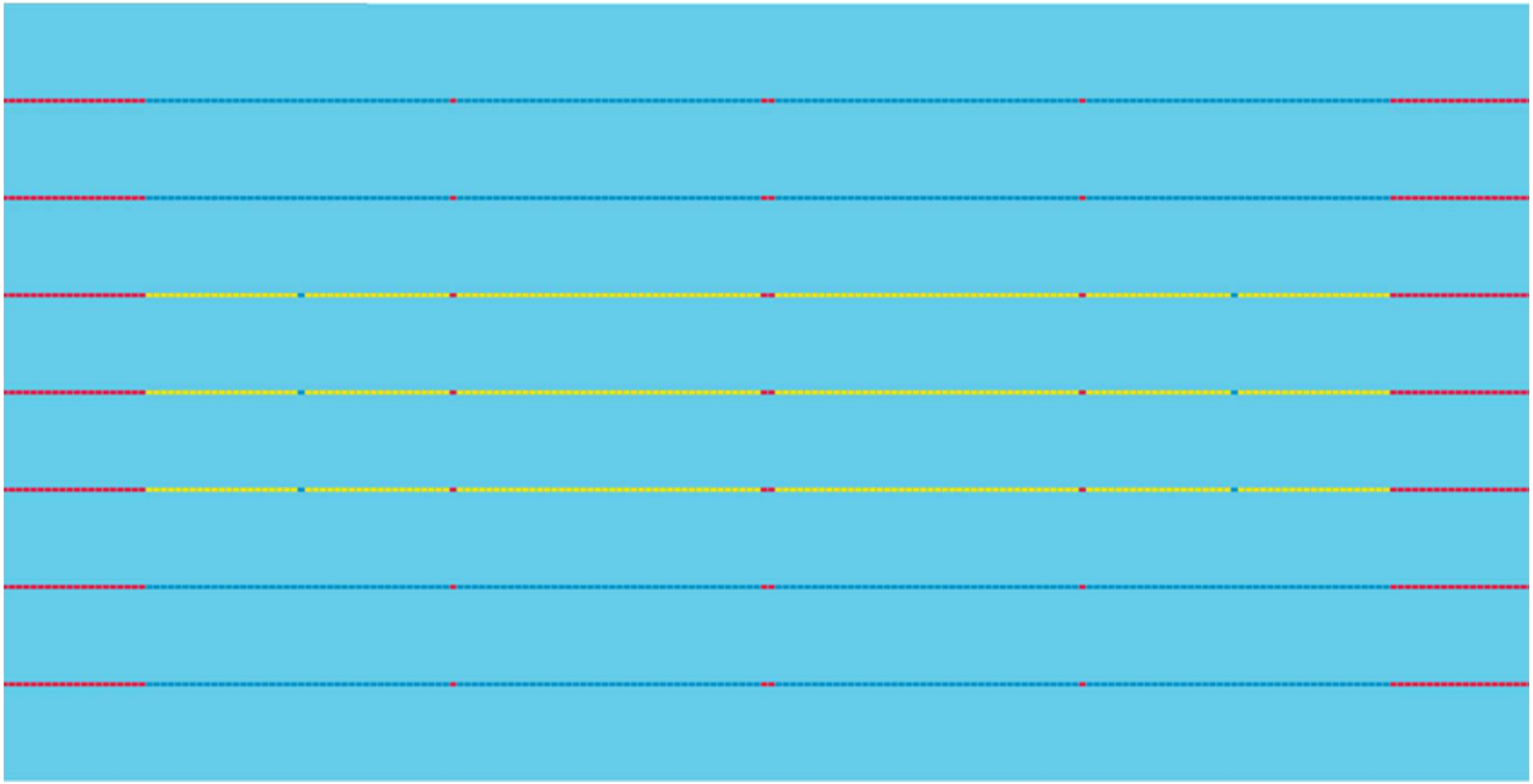

ad Decision or Merge
[condition is true]
Action on True
Decision Node
Merge Node
Action of False
[condition is false]

An activity diagram using merge notation

- In this diagram :
- we can’t have both sides executing concurrently, but they finally merge into one.
- A number can’t be both odd and even at the same time.

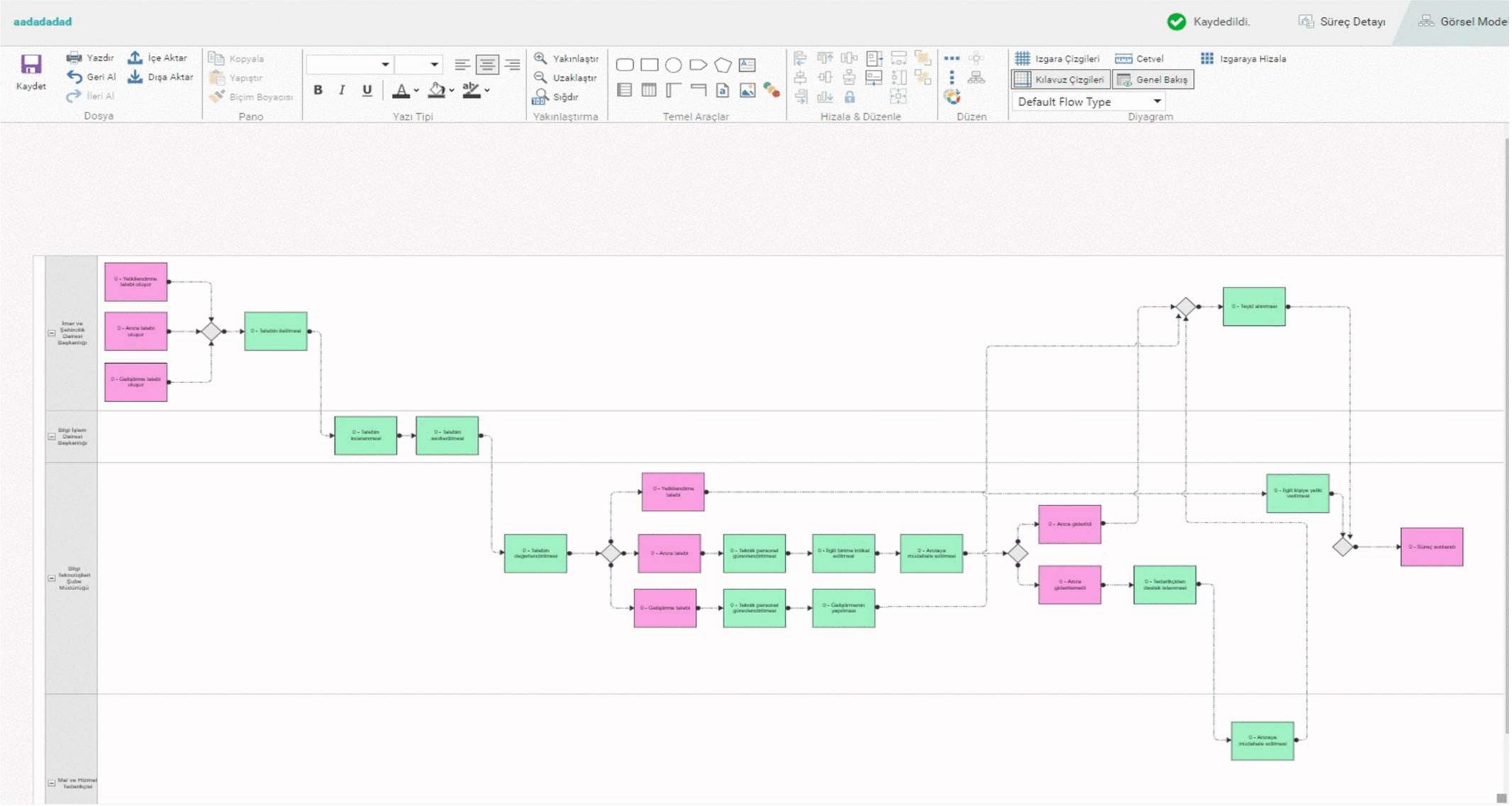

is the number odd
Yes
NO
add 5 to the number
add 7 to the number
Multiply it by 5
Multiply it by 7
Display the number
Display the number
add 20 to the number

OLYMPIC-SIZED SWIMMING POOL



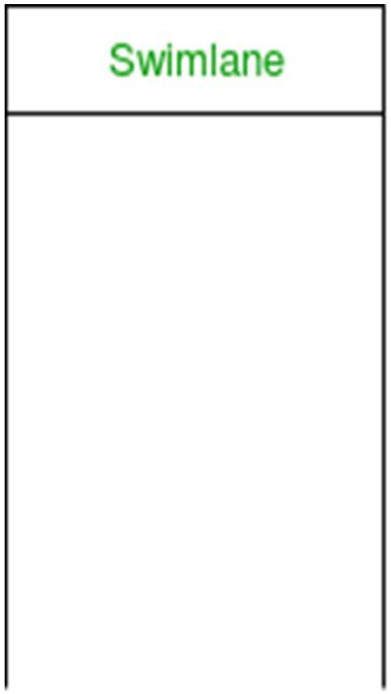
Swimlanes

- We use swimlanes for grouping related activities in one column.
- Swimlanes group related activities into one column or one row.
- Swimlanes can be vertical and horizontal.
- Swimlanes are used to add modularity to the activity diagram.
- It is not mandatory to use swimlanes.
- They usually give more clarity to the activity diagram.
- It’s similar to creating a function in a program.
- It’s not mandatory to do so, but, it is a recommended practice.


aadadadad
Kaydedildi.
Süreç Detayı
Görsel Mode
Kaydet
Yazdır
Geri Al
İleri Al
İçe Aktar
Dışa Aktar
Dosya
Kopyala
Yapıştır
Biçim Boyacısı
Pano
B
I
U
Yazı Tipi
Yakınlaştır
Uzaklaştır
Sığdır
Yakınlaştırma
Temel Araçlar
Hizala & Düzenle
Düzen
Izgara Çizgileri
Cetvel
Izgaraya Hizala
Kılavuz Çizgileri
Genel Bakış
Default Flow Type
Diyagram
İmar ve Şehircilik Dairesi Başkanlığı
Bilgi İşlem Dairesi Başkanlığı
Bilgi Teknolojileri Şube Müdürlüğü
Mal ve Hizmet Tedarikçisi
0 - Yetkilendirme talebi oluşur
0 - Arıza talebi oluşur
0 - Geliştirme talebi oluşur
0 - Talebin iletilmesi
0 - Talebin incelenmesi
0 - Talebin sevkedilmesi
0 - Talebin değerlendirilmesi
0 - Yetkilendirme talebi
0 - Arıza talebi
0 - Geliştirme talebi
0 - Teknik personel görevlendirilmesi
0 - İlgili birime iletilmesi
0 - Arızaya müdahale edilmesi
0 - Teknik personel görevlendirilmesi
0 - Geliştirmenin yapılması
0 - Arıza giderildi
0 - Arıza giderilemedi
0 - Tedarikçiden destek istenmesi
0 - Teyid alınması
0 - İlgili kişiye yetki verilmesi
0 - Süreç sonlandı
0 - Arızaya müdahale edilmesi

swimlanes notation

- We use a rectangular column to represent a swimlane

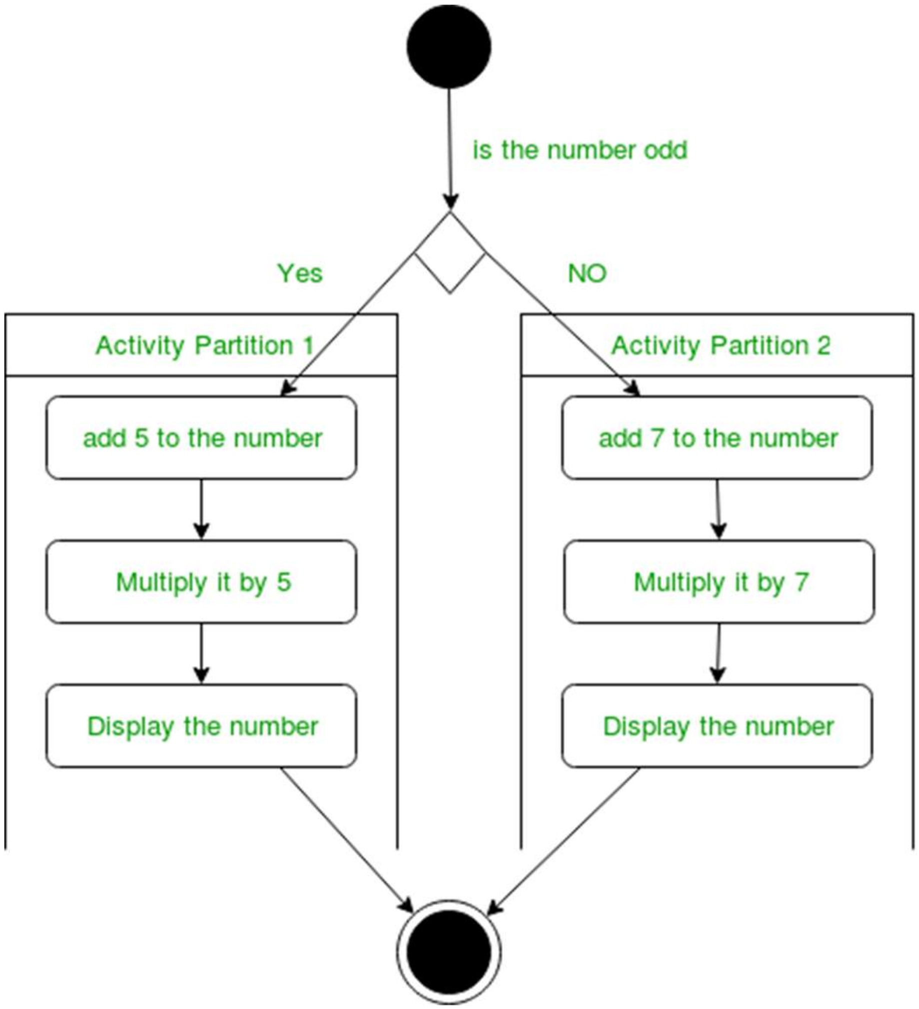


For example

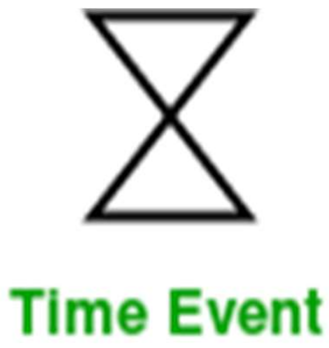
- Here different set of activities are executed based on if the number is odd or even.

- These activities are grouped into a swimlane.

- an activity diagram making use of swimlanes


is the number odd
Yes
NO
Activity Partition 1
Activity Partition 2
add 5 to the number
add 7 to the number
Multiply it by 5
Multiply it by 7
Display the number
Display the number

Time Event



time event notation

We can have a scenario where an event takes some time to complete.

We use an hourglass to represent a time event.

For example

- Let us assume that the processing of an image takes a lot of time.
- Then it can be represented as.

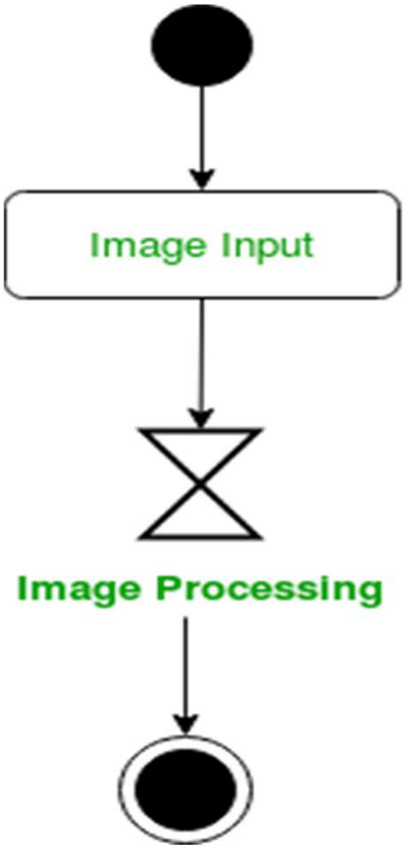


Image Input
Image Processing

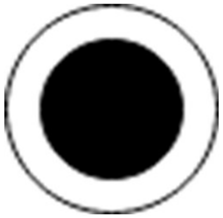
- an activity diagram using time event

Final State or End State

- The state which the system reaches when a particular process or activity ends is known as a Final State or End State.

- We use a filled circle within a circle notation to represent the final state in a state machine diagram.

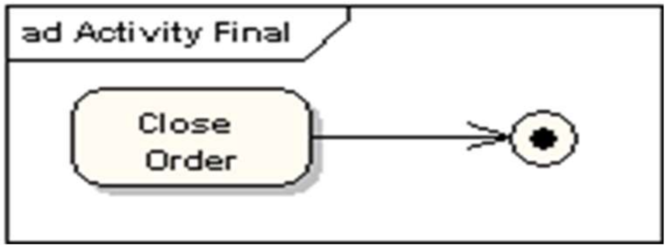
- A system or a process can have multiple final states.



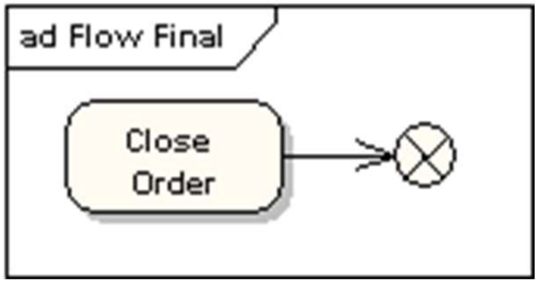
notation for final state

Final Node

- There are two types of final node: activity and flow final nodes. The activity final node is depicted as a circle with a dot inside.


ad Activity Final
Close Order

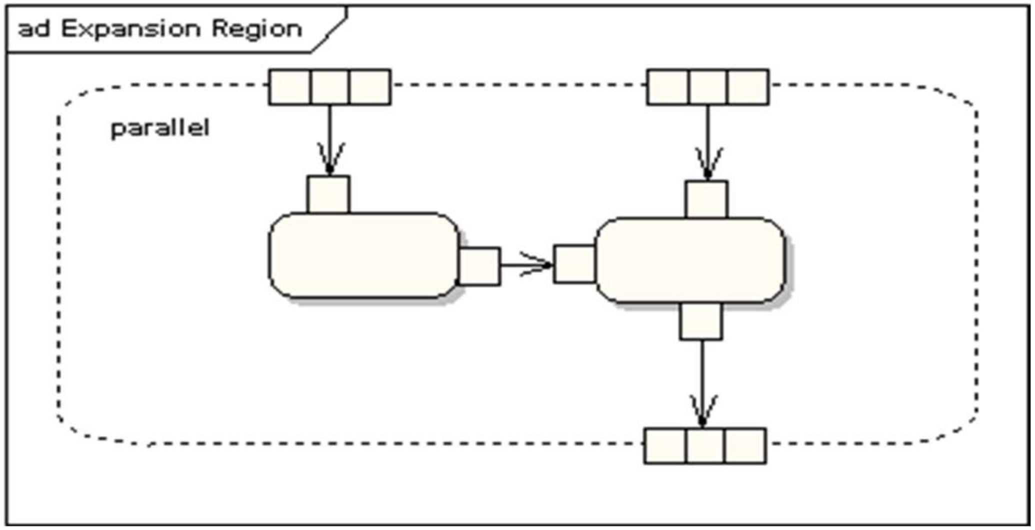
The flow final node is depicted as a circle with a cross inside.



The difference between the two node types is that the flow final node denotes the end of a single control flow; the activity final node denotes the end of all control flows within the activity.

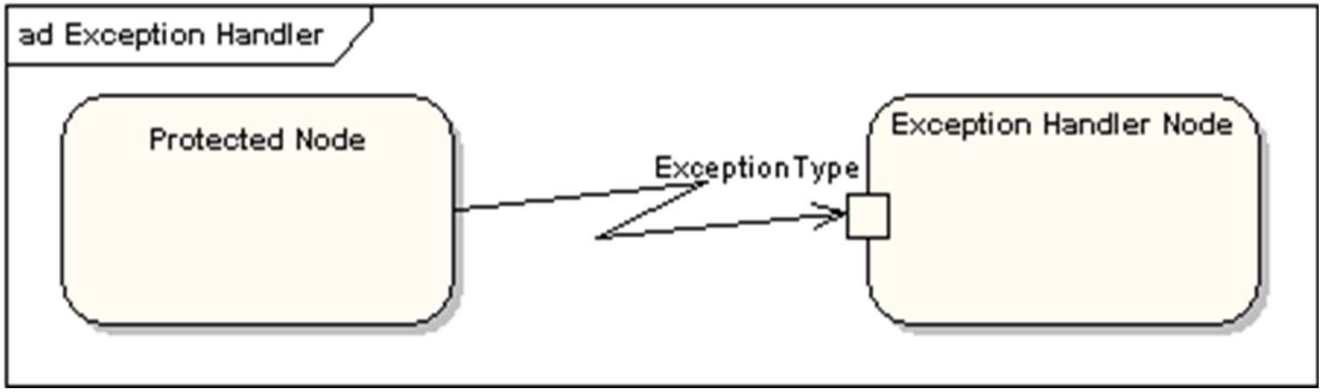
Expansion Region

- An expansion region is a structured activity region that executes multiple times.
- Input and output expansion nodes are drawn as a group of three boxes representing a multiple selection of items.
- The keyword "iterative", "parallel" or "stream" is shown in the top left corner of the region.



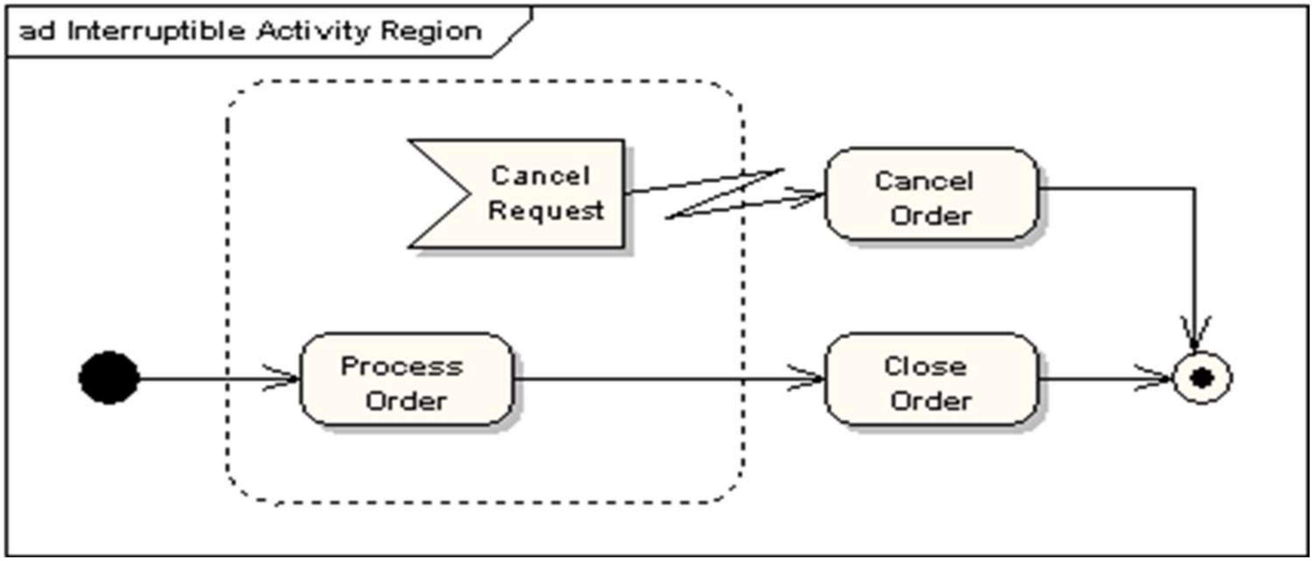
Exception Handlers

- Exception Handlers can be modelled on activity diagrams as in the example below.


ad Exception Handler
Protected Node
ExceptionType
Exception Handler Node

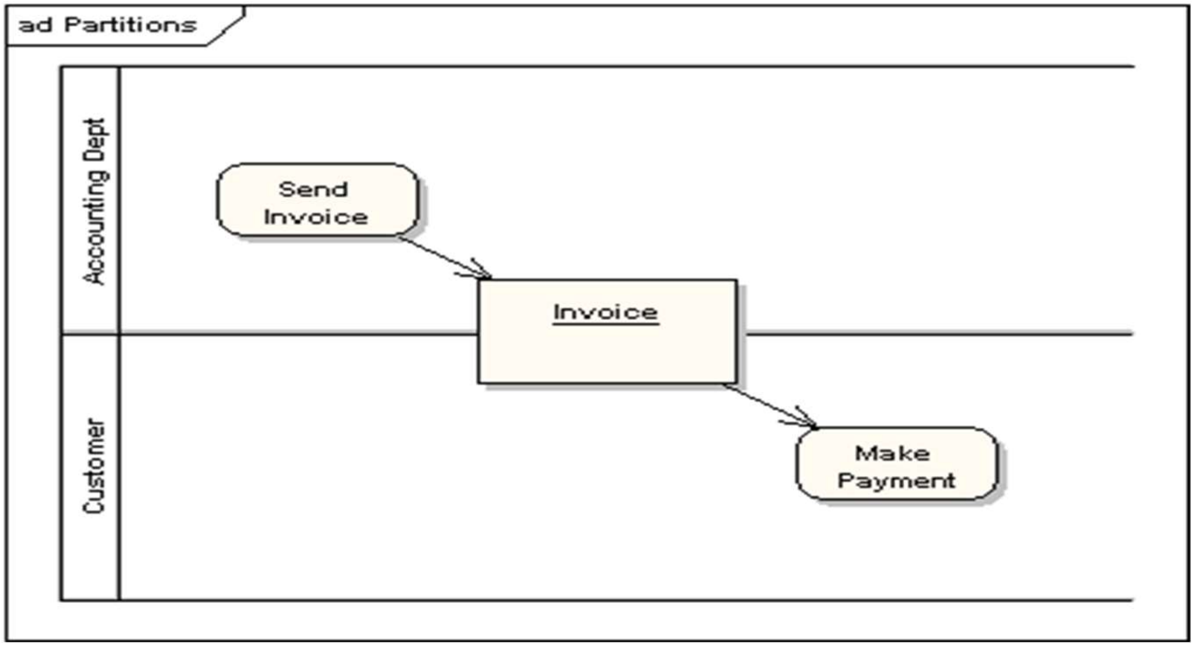
Interruptible Activity Region

- An interruptible activity region surrounds a group of actions that can be interrupted.
- In the very simple example below, the "Process Order" action will execute until completion, when it will pass control to the "Close Order" action, unless a "Cancel Request" interrupt is received, which will pass control to the "Cancel Order" action.


ad Interruptible Activity Region
Cancel Request
Cancel Order
Process Order
Close Order

Partition

- An activity partition is shown as either a horizontal or vertical swimlane.
- In the following diagram, the partitions are used to separate actions within an activity into those performed by the accounting department and those performed by the customer.


ad Partitions
Accounting Dept
Customer
Send Invoice
Invoice
Make Payment

How to Draw an activity diagram ?

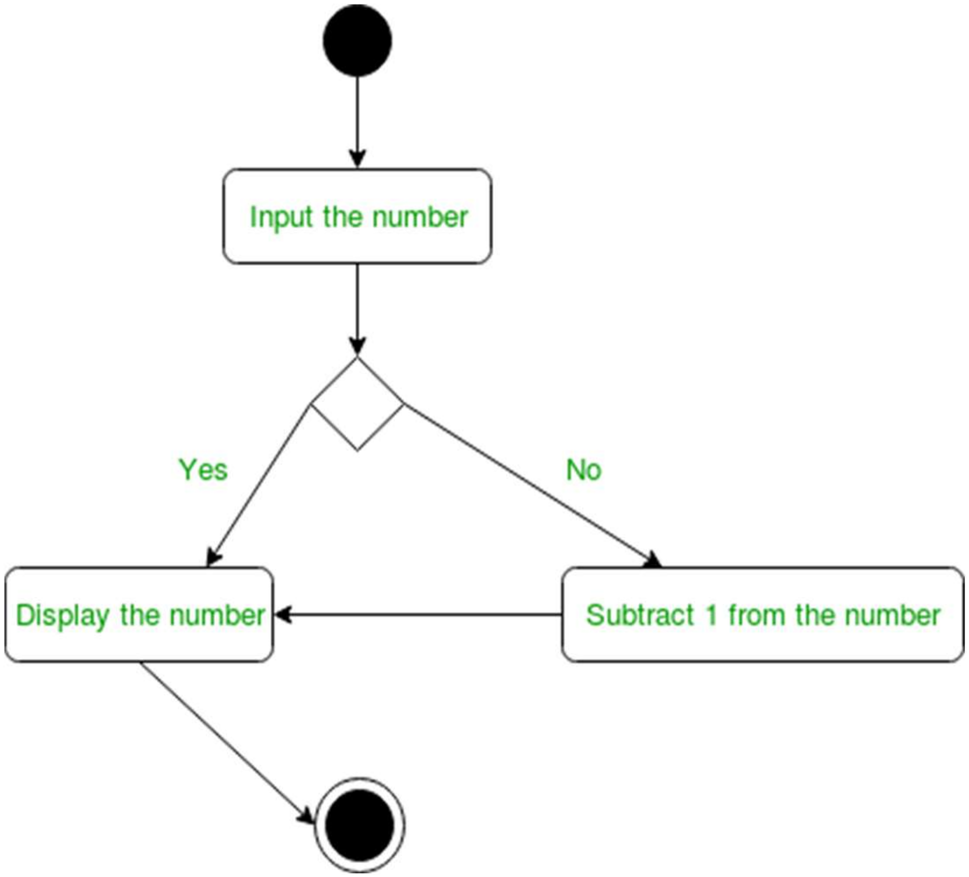
1. Identify the initial state and the final states.

2. Identify the intermediate activities needed to reach the final state from the initial state.

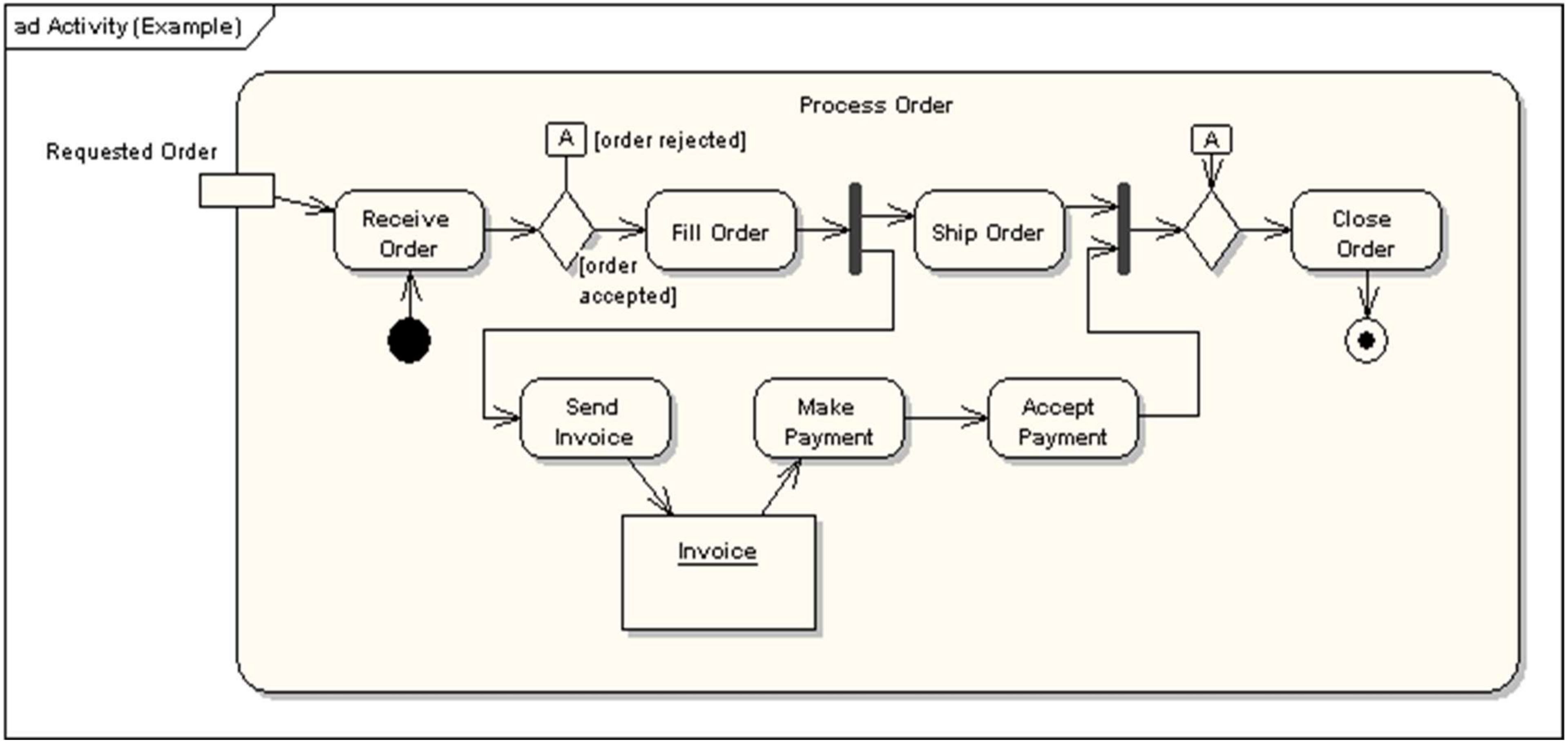
3. Identify the conditions or constraints which cause the system to change control flow.

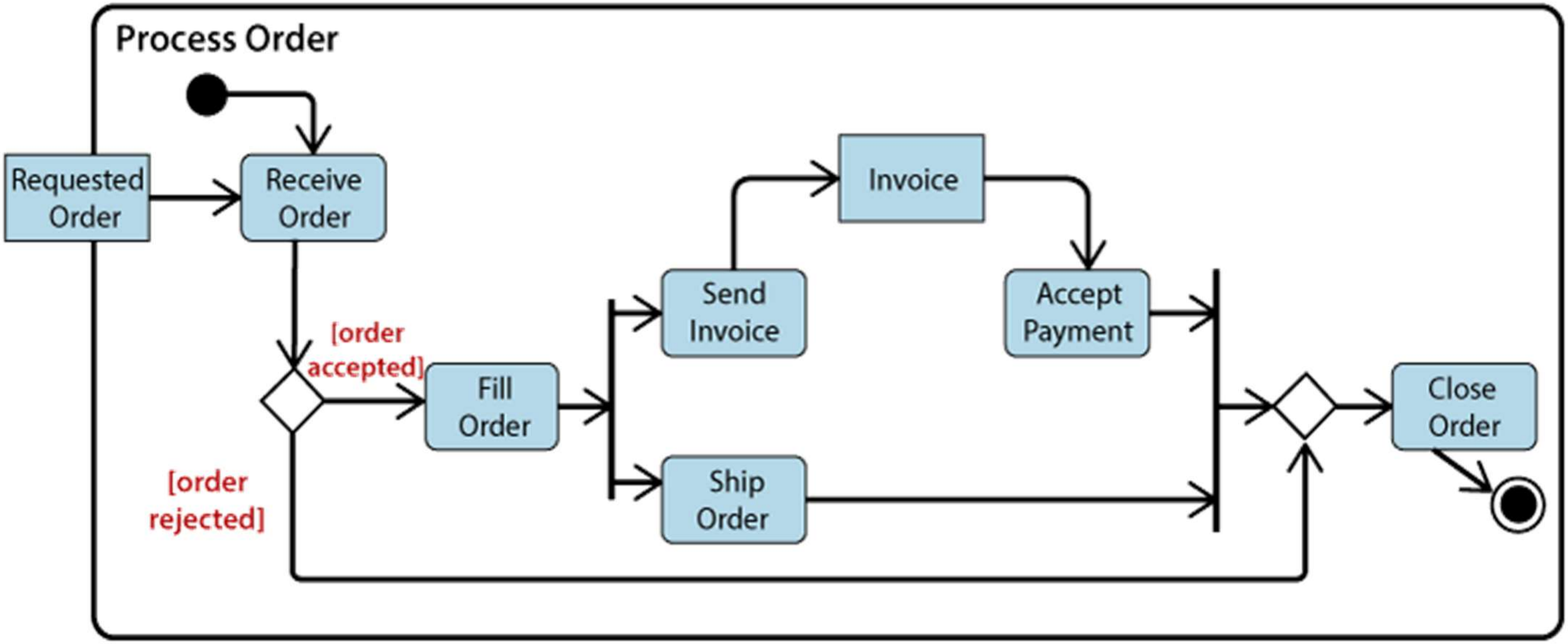
4. Draw the diagram with appropriate notations.

an activity diagram


Input the number
Yes
No
Display the number
Subtract 1 from the number

The above diagram prints the number if it is odd otherwise it subtracts one from the number and displays it.


ad Activity (Example)
Process Order
Requested Order
Receive Order
A
[order rejected]
[order accepted]
Fill Order
Ship Order
A
Close Order
Send Invoice
Make Payment
Accept Payment
Invoice


Process Order
Requested Order
Receive Order
[order accepted]
[order rejected]
Fill Order
Send Invoice
Invoice
Accept Payment
Ship Order
Close Order

Uses of an Activity Diagram

- Dynamic modelling of the system or a process.
- Illustrate the various steps involved in a UML use case.
- Model software elements like methods, operations and functions.
- We can use Activity diagrams to depict concurrent activities easily.
- Show the constraints, conditions and logic behind algorithms.

When to use an Activity Diagram?

- An activity diagram can be used to portray business processes and workflows. Also, it used for modeling business as well as the software. An activity diagram is utilized for the followings:

1. To graphically model the workflow in an easier and understandable way.
2. To model the execution flow among several activities.
3. To model comprehensive information of a function or an algorithm employed within the system.
4. To model the business process and its workflow.
5. To envision the dynamic aspect of a system.
6. To generate the top-level flowcharts for representing the workflow of an application.
7. To represent a high-level view of a distributed or an object-oriented system.

Unified Modeling Language (UML) State Diagrams

State diagram

- state diagram is used to represent the condition of the system or part of the system at finite instances of time.

- It’s a behavioral diagram and it represents the behavior using finite state transitions.

- State diagrams are also referred to as State machines and State-chart Diagrams.

- These terms are often used interchangeably.

- So simply, a state diagram is used to model the dynamic behavior of a class in response to time and changing external stimuli.

- We can say that each and every class has a state but we don’t model every class using State diagrams.

- We prefer to model the states with three or more states.

Uses of state chart diagram

- We use it to state the events responsible for change in state
- (we do not show what processes cause those events).
- We use it to model the dynamic behavior of the system .
- To understand the reaction of objects/classes to internal or external stimuli.

First let us understand what are Behavior diagrams?

- There are two types of diagrams in UML :

1. Structure Diagrams – Used to model the static structure of a system, for example- class diagram, package diagram, object diagram, deployment diagram etc.

2. Behavior diagram – Used to model the dynamic change in the system over time.

- They are used to model and construct the functionality of a system.

- So, a behavior diagram simply guides us through the functionality of the system using Use case diagrams, Interaction diagrams, Activity diagrams and State diagrams.

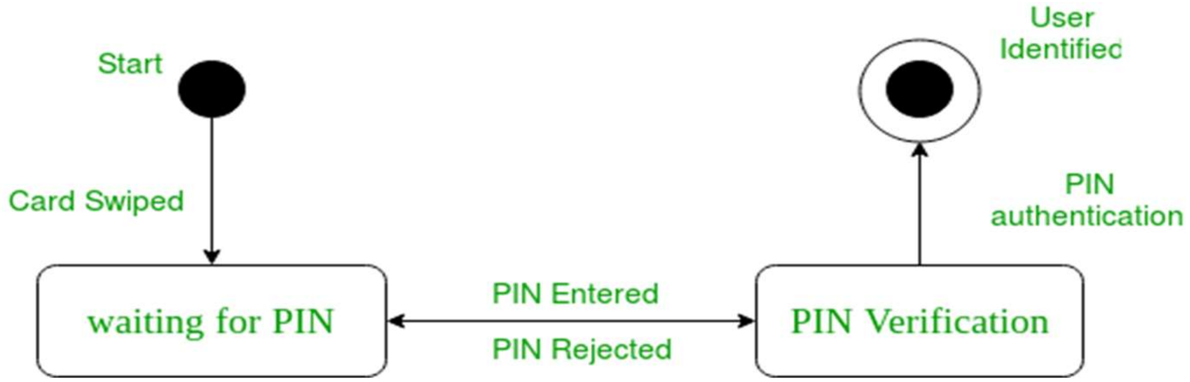
Difference between state diagram and flowchart

- The basic purpose of a state diagram is to portray various changes in state of the class and not the processes or commands causing the changes.

- However, a flowchart on the other hand portrays the processes or commands that on execution change the state of class or an object of the class.

  - a state diagram for user verification

  - The state diagram shows the different states in which the verification sub-system or class exist for a particular system.


Start
Card Swiped
waiting for PIN
PIN Entered
PIN Rejected
PIN Verification
PIN authentication
User Identified

State Machine vs. Flowchart

State Machine	Flowchart
It portrays several states of a system.	It demonstrates the execution flow of a program.
It encompasses the concept of WAIT, i.e., wait for an event or an action.	It does not constitute the concept of WAIT.
It is for real-world modeling systems.	It envisions the branching sequence of a system.
It is a modeling diagram.	It is a data flow diagram (DFD)
It is concerned with several states of a system.	It focuses on control flow and path.

1. Behavioral state machine
   The behavioral state machine diagram records the behavior of an object within the system.

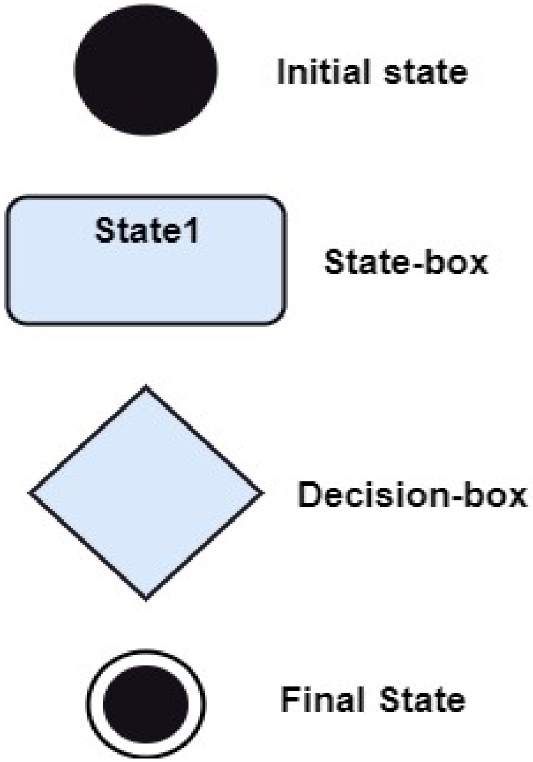
- It depicts an implementation of a particular entity.
- It models the behavior of the system.

2. Protocol state machine
   It captures the behavior of the protocol.

- The protocol state machine depicts the change in the state of the protocol and parallel changes within the system.
- But it does not portray the implementation of a particular component.

Notation of a State Machine Diagram

- Following are the notations of a state machine diagram enlisted below:


Initial state
State1
State-box
Decision-box
Final State

Notation of a State Machine Diagram

1. Initial state: It defines the initial state (beginning) of a system, and it is represented by a black filled circle.

2. Final state: It represents the final state (end) of a system. It is denoted by a filled circle present within a circle.

3. Decision box: It is of diamond shape that represents the decisions to be made on the basis of an evaluated guard.

4. Transition: A change of control from one state to another due to the occurrence of some event is termed as a transition.

5. It is represented by an arrow labeled with an event due to which the change has ensued.

6. State box: It depicts the conditions or circumstances of a particular object of a class at a specific point of time.

7. A rectangle with round corners is used to represent the state box.

Types of State

- The UML consist of three states:

1. **Simple state:** It does not constitute any substructure.
2. **Composite state:** It consists of nested states (substates), such that it does not contain more than one initial state and one final state. It can be nested to any level.
3. **Submachine state:** The submachine state is semantically identical to the composite state, but it can be reused.

How to Draw a State Machine Diagram?

- The state machine diagram is used to portray various states underwent by an object.
- The change in one state to another is due to the occurrence of some event.
- All of the possible states of a particular component must be identified before drawing a state machine diagram.
- The primary focus of the state machine diagram is to depict the states of a system.
- These states are essential while drawing a state transition diagram.
- The objects, states, and events due to which the state transition occurs must be acknowledged before the implementation of a state machine diagram.

Following are the steps that are to be incorporated while drawing a state machine diagram:

1. A unique and understandable name should be assigned to the state transition that describes the behavior of the system.

2. Out of multiple objects, only the essential objects are implemented.

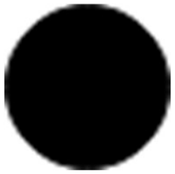
3. A proper name should be given to the events and the transitions.

State machine diagram is used for:

1. For modeling the object states of a system.
2. For modeling the reactive system as it consists of reactive objects.
3. For pinpointing the events responsible for state transitions.
4. For implementing forward and reverse engineering.

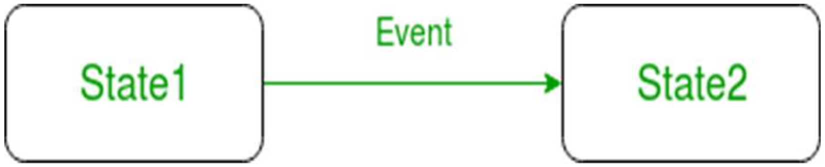
Basic components of a state chart diagram

1. **Initial state –** We use a black filled circle represent the initial state of a System or a class.



initial state notation

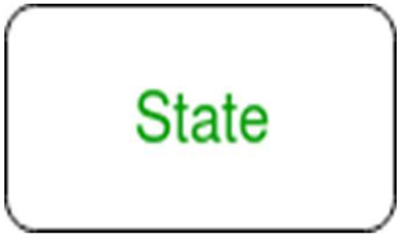
- **Transition –** We use a solid arrow to represent the transition or change of control from one state to another.
- The arrow is labelled with the event which causes the change in state.



transition

- State – We use a rounded rectangle to represent a state.

- A state represents the conditions or circumstances of an object of a class at an instant of time.

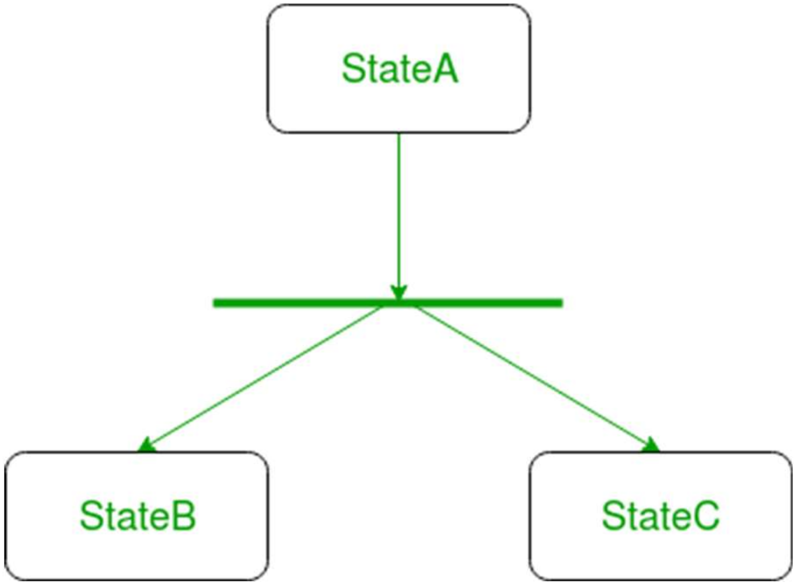


state notation

Fork

- **Fork** – We use a rounded solid rectangular bar to represent a Fork notation with incoming arrow from the parent state and outgoing arrows towards the newly created states.

- We use the fork notation to represent a state splitting into two or more concurrent states.

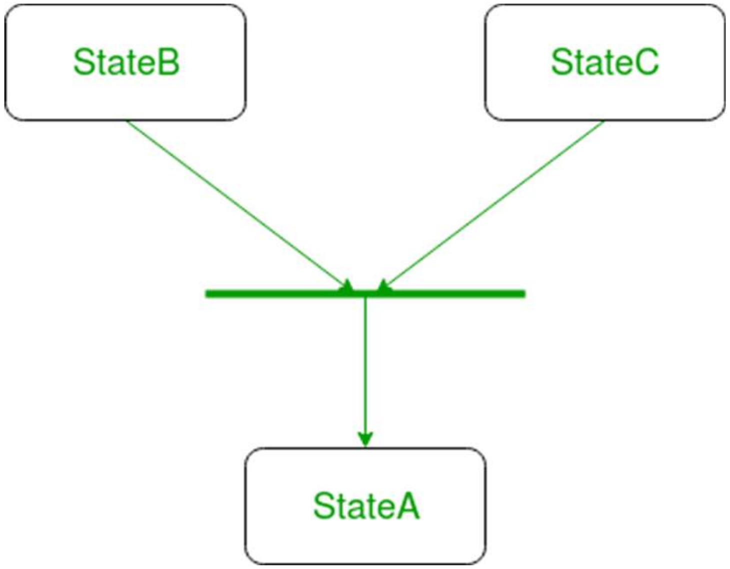


a diagram using the fork notation

Join

- **Join** – We use a rounded solid rectangular bar to represent a Join notation with incoming arrows from the joining states and outgoing arrow towards the common goal state.

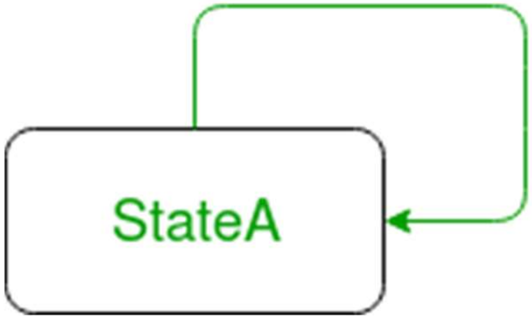
- We use the join notation when two or more states concurrently converge into one on the occurrence of an event or events.



a diagram using join notation

Self transition

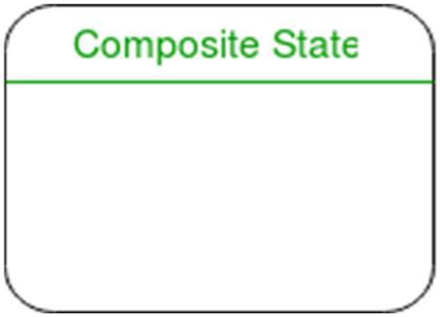
- **Self transition** – We use a solid arrow pointing back to the state itself to represent a self transition.
- There might be scenarios when the state of the object does not change upon the occurrence of an event.
- We use self transitions to represent such cases.



self transition notation

Composite state

- **Composite state –** We use a rounded rectangle to represent a composite state also.
- We represent a state with internal activities using a composite state.

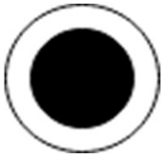


a state with internal activities

Final state

- Final state – We use a filled circle within a circle notation to represent the final state in a state machine diagram.

final state notation

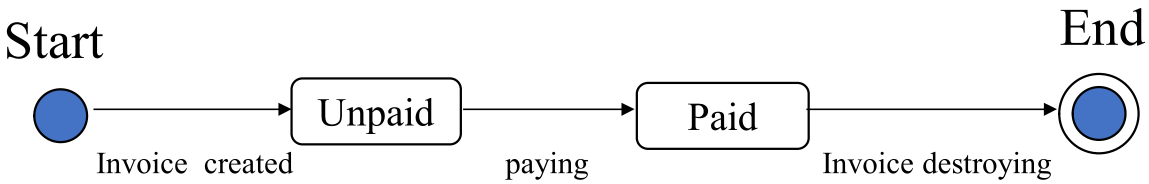


Steps to draw a state diagram

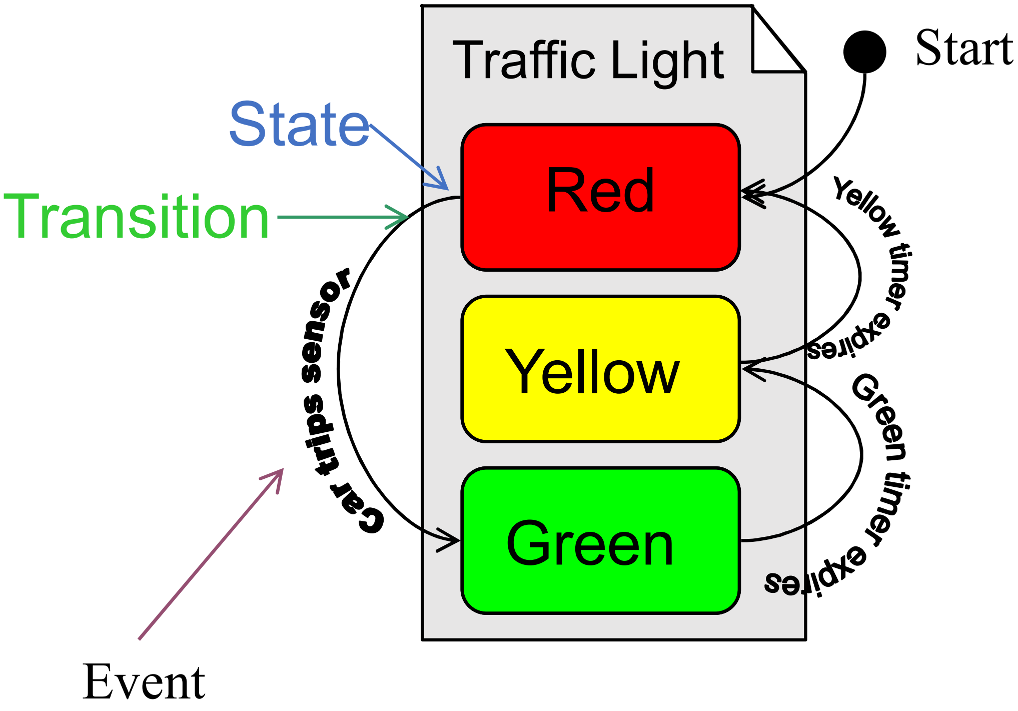
1. Identify the initial state and the final terminating states.
2. Identify the possible states in which the object can exist (boundary values corresponding to different attributes guide us in identifying different states).
3. Label the events which trigger these transitions.

State Diagrams (Billing Example)

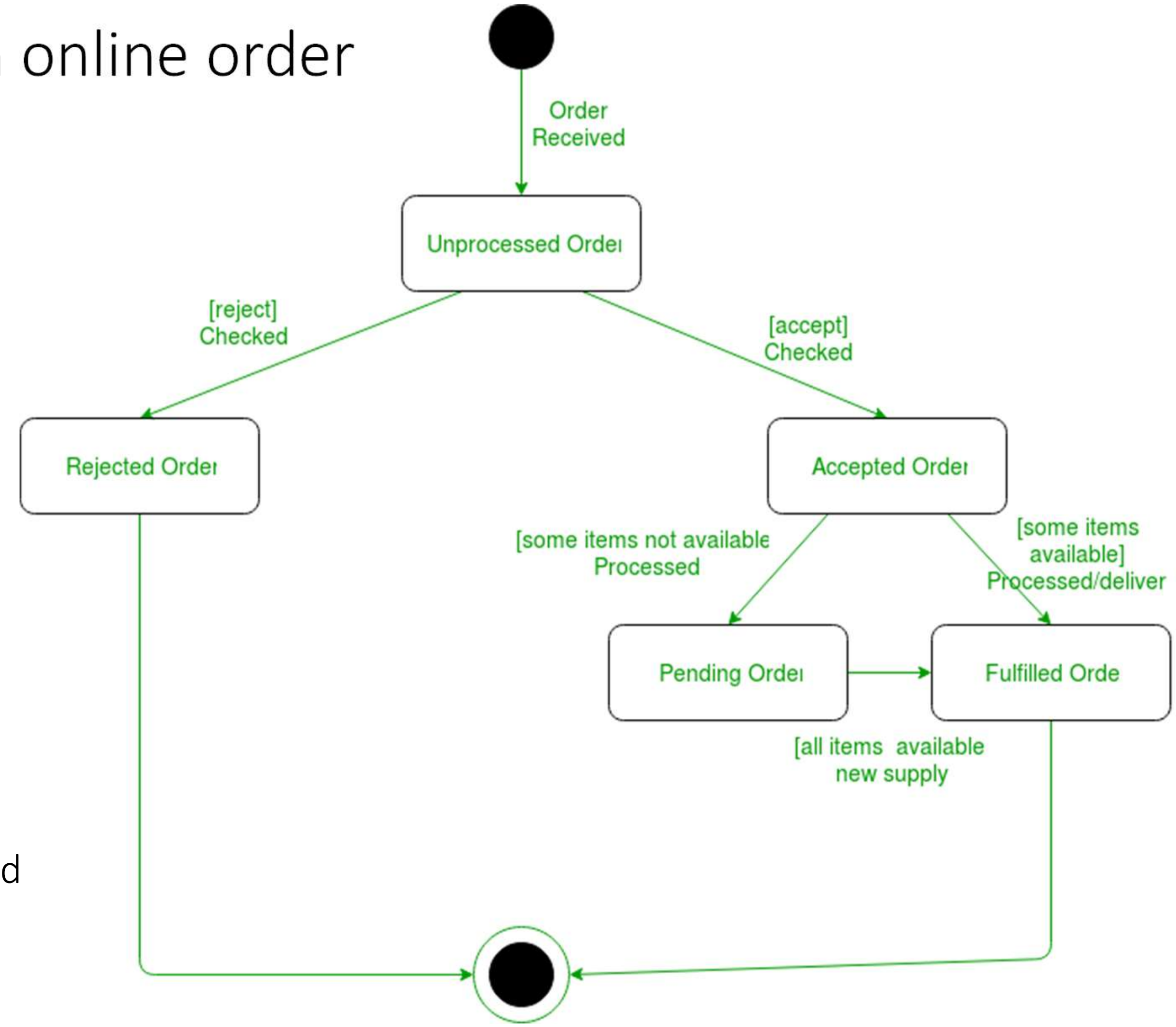
State Diagrams show the sequences of states an object goes through during its life cycle in response to stimuli, together with its responses and actions; an abstraction of all possible behaviors.


Start
End
Unpaid
Paid
Invoice created
paying
Invoice destroying

State Diagrams (Traffic light example)


Traffic Light
Start
State
Transition
Red
Yellow
Green
Yellow timer expires
Green timer expires
Car trips sensor
Event

state diagram for an online order


Order Received
Unprocessed Orde
[reject] Checked
[accept] Checked
Rejected Orde
Accepted Orde
[some items not available Processed
[some items available] Processed/deliver
Pending Orde
Fulfilled Orde
[all items available new supply

Note – Here we could have also treated fulfilled order and rejected order as final states separately.

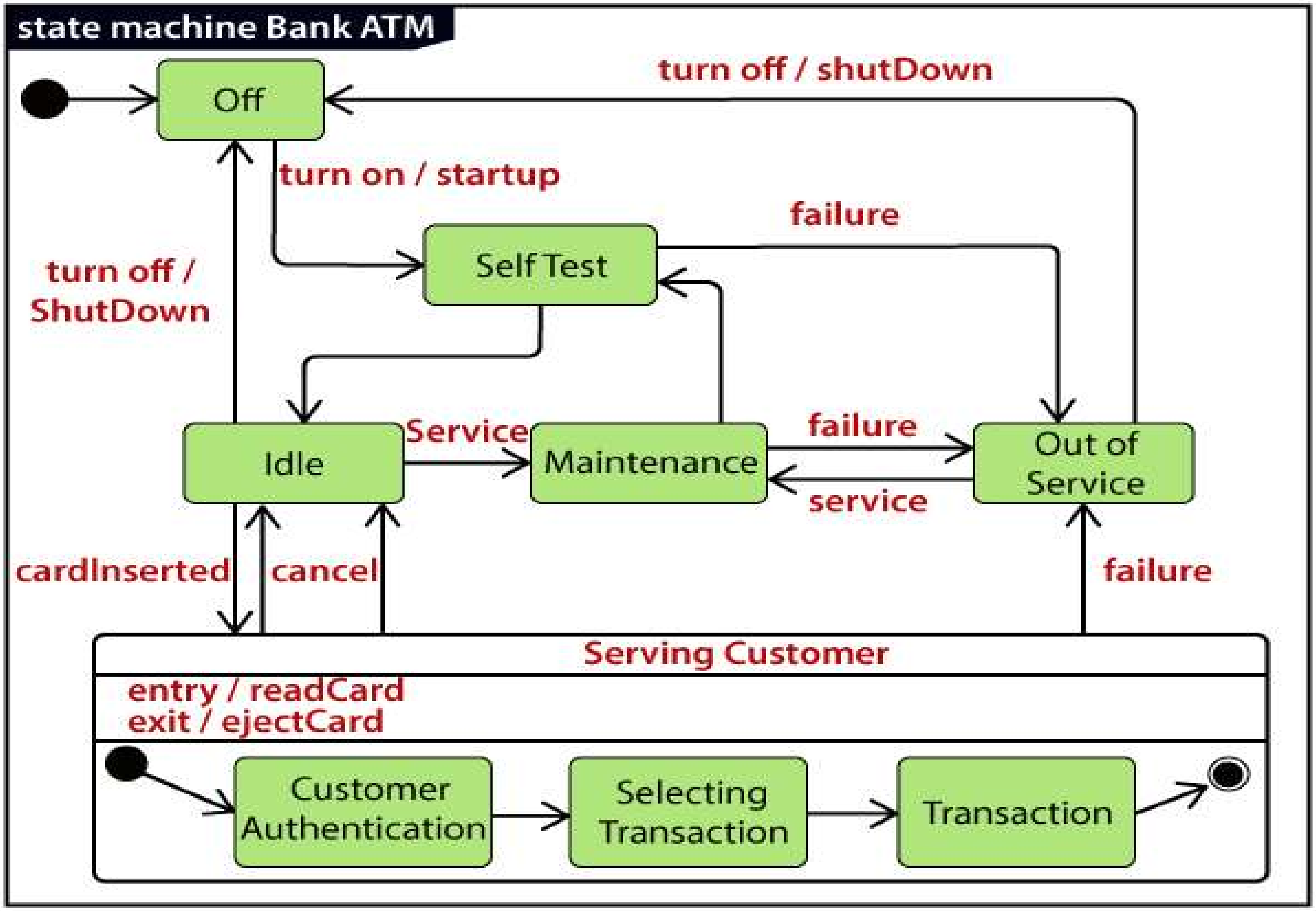
The UML diagrams we draw depend on the system we aim to represent.

Here is just an example of how an online ordering system might look like :

1. On the event of an order being received, we transit from our initial state to Unprocessed order state.
2. The unprocessed order is then checked.
3. If the order is rejected, we transit to the Rejected Order state.
4. If the order is accepted and we have the items available we transit to the fulfilled order state.
5. However if the items are not available we transit to the Pending Order state.
6. After the order is fulfilled, we transit to the final state.
7. In this example, we merge the two states i.e. Fulfilled order and Rejected order into one final state.

Example of a State Machine Diagram

- An example of a top-level state machine diagram showing Bank Automated Teller Machine (ATM) is given below.
- Initially, the ATM is turned off. After the power supply is turned on, the ATM starts performing the startup action and enters into the Self Test state.
- If the test fails, the ATM will enter into the Out Of Service state, or it will undergo a triggerless transition to the Idle state. This is the state where the customer waits for the interaction.
- Whenever the customer inserts the bank or credit card in the ATM's card reader, the ATM state changes from Idle to Serving Customer, the entry action readCard is performed after entering into Serving Customer state.
- Since the customer can cancel the transaction at any instant, so the transition from Serving Customer state back to the Idle state could be triggered by cancel event.


state machine Bank ATM
Off
turn off / shutDown
turn on / startup
failure
Self Test
turn off / ShutDown
Idle
Service
Maintenance
failure
service
Out of Service
cardInserted
cancel
failure
Serving Customer
entry / readCard
exit / ejectCard
Customer Authentication
Selecting Transaction
Transaction

Example of a State Machine Diagram

- Here the Serving Customer is a composite state with sequential substates that are Customer Authentication, Selecting Transaction, and Transaction.

- Customer Authentication and Transaction are the composite states itself is displayed by a hidden decomposition indication icon.

- After the transaction is finished, the Serving Customer encompasses a triggerless transition back to the Idle state.

- On leaving the state, it undergoes the exit action ejectCard that discharges the customer card.

Lists : State and Stimulus

- Example: A course in a registration system

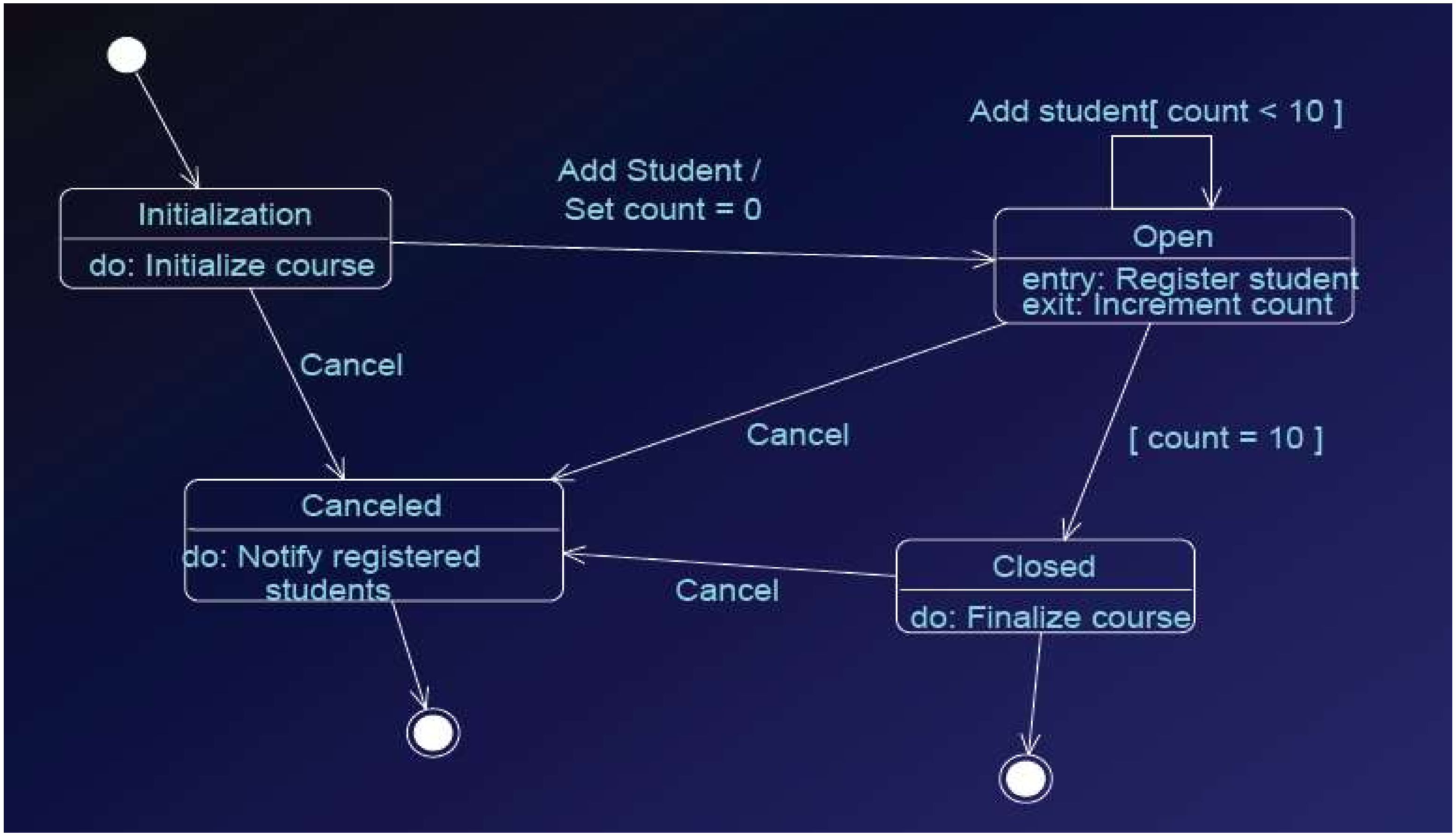
State List

State	Description
Initialized	Exists, but no details set up
Open	Ready for registration
Canceled	
Closed	
Deleted	

Stimulus List

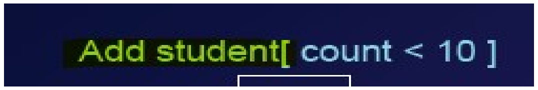
Stimulus	Description
Create course	Initializes the course
Register students	Happens inside the open state.
Cancel the course	Cancels an existing course but a record of the course existing remains. Tell students it cancels
Delete a course	Delete a mistakenly created course
Fill a course	Closes the course
Notify about cancelation	Happens inside the canceled state

Course State Diagram

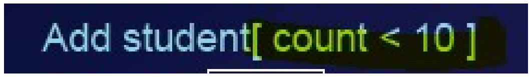

Add student[count < 10]
Add Student /
Set count = 0
Initialization
do: Initialize course
Open
entry: Register student
exit: Increment count
Cancel
Cancel
[count = 10]
Canceled
do: Notify registered students
Cancel
Closed
do: Finalize course

Words on the Line:

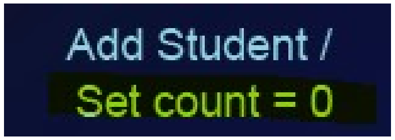
- Trigger : an action


Add student[count < 10]

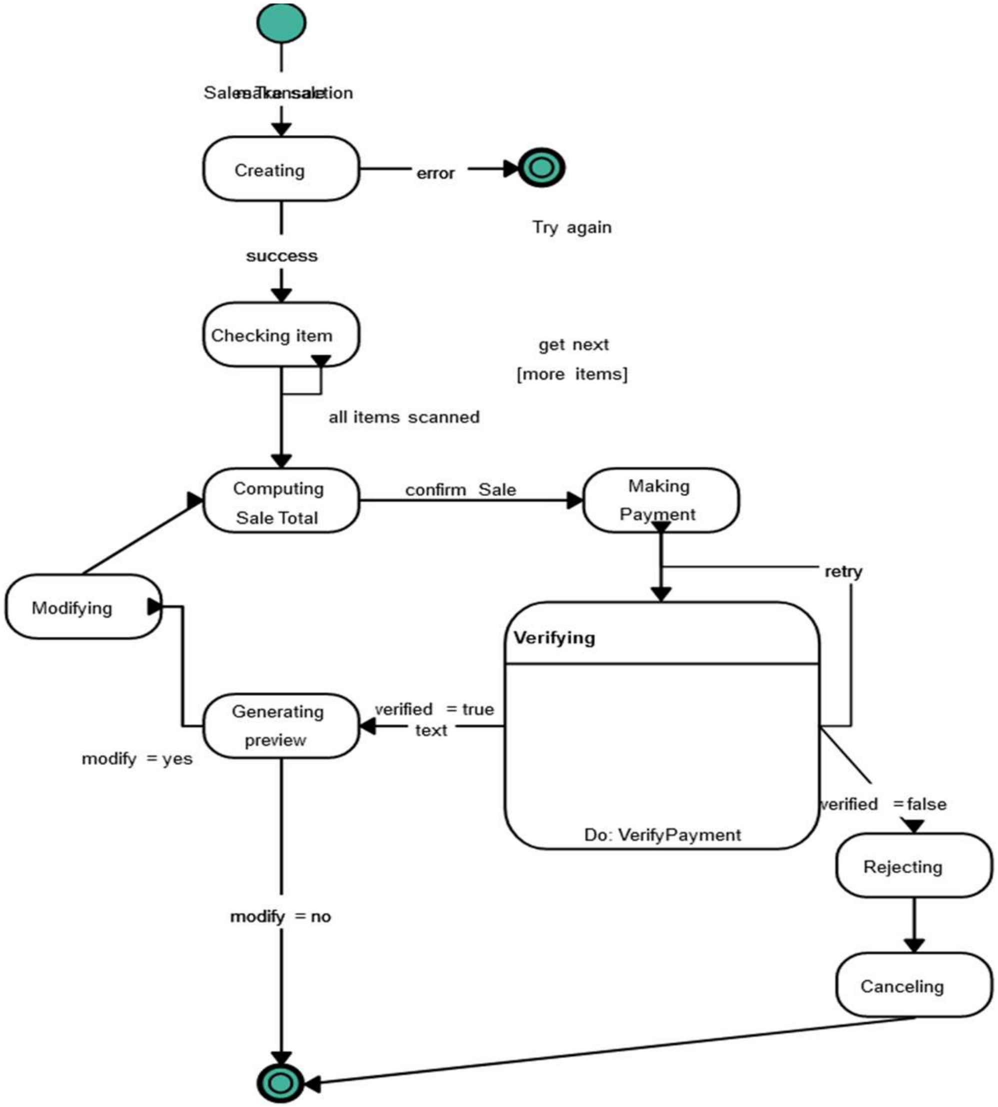
- Guard (inside []) : a condition

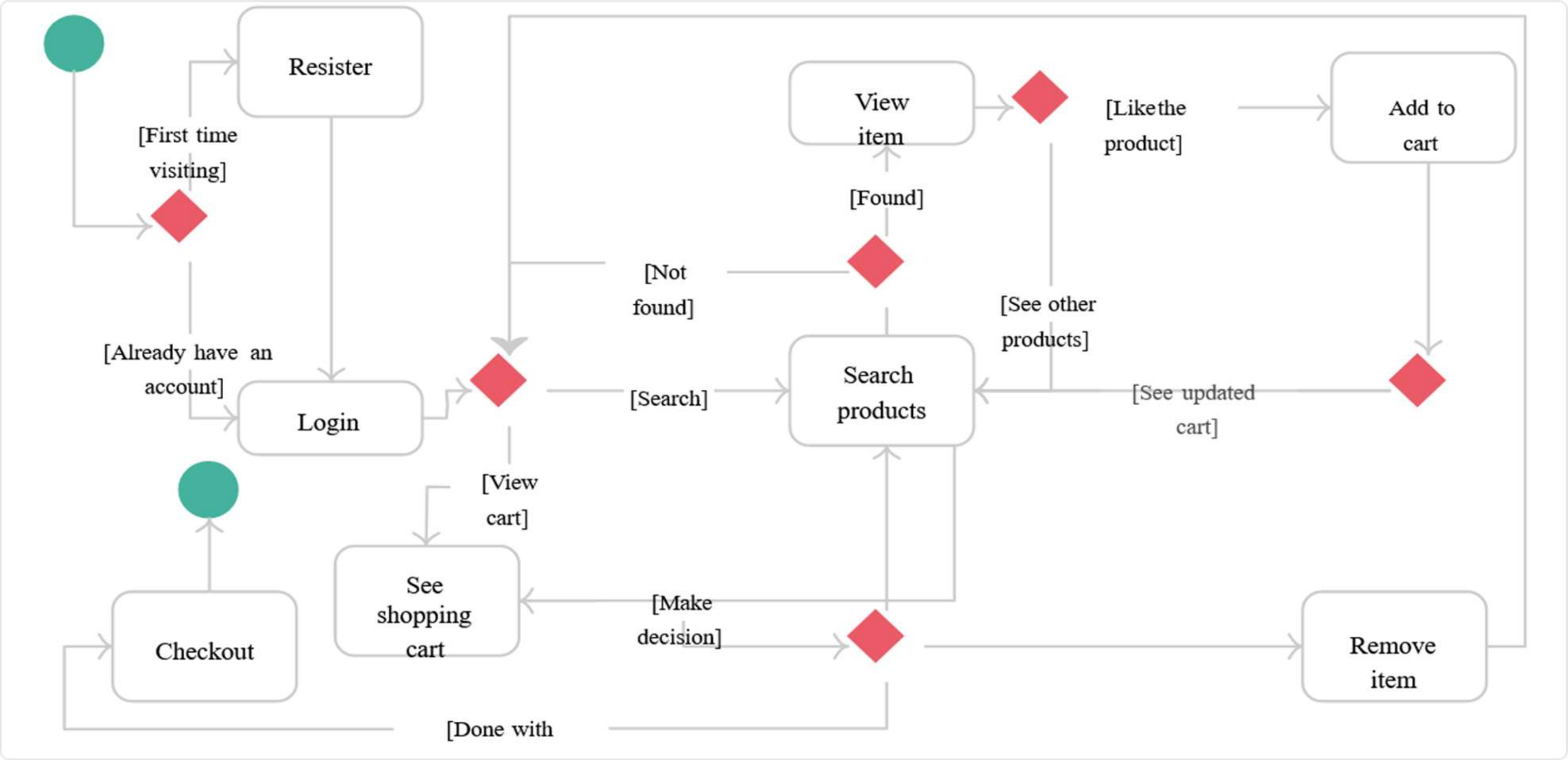

Add student[count < 10]

- Transitional behavior (after /) : a message or variable setting


Add Student /
Set count = 0

POS System State Diagram


Sales Transaction
make sale
Creating
error
Try again
success
Checking item
get next
[more items]
all items scanned
Computing
Sale Total
confirm Sale
Making
Payment
retry
Modifying
Verifying
verified = true
text
Generating
preview
modify = yes
verified = false
Do: VerifyPayment
Rejecting
modify = no
Canceling


Resister
[First time visiting]
[Already have an account]
Login
View item
[Like the product]
Add to cart
[Found]
[Not found]
[See other products]
[Search]
Search products
[See updated cart]
[View cart]
See shopping cart
Checkout
[Make decision]
Remove item
[Done with

Online Shopping State Diagram

Summary

- Start with states
- List of state and stimulus helps you think of all different states
  - Some actions may happen inside states and not be stimulus for movement
- Draw rounded boxes for states
- Look at each stimulus to see if it gets an arrow from that state.
- Not an activity diagram

UML diagram summary

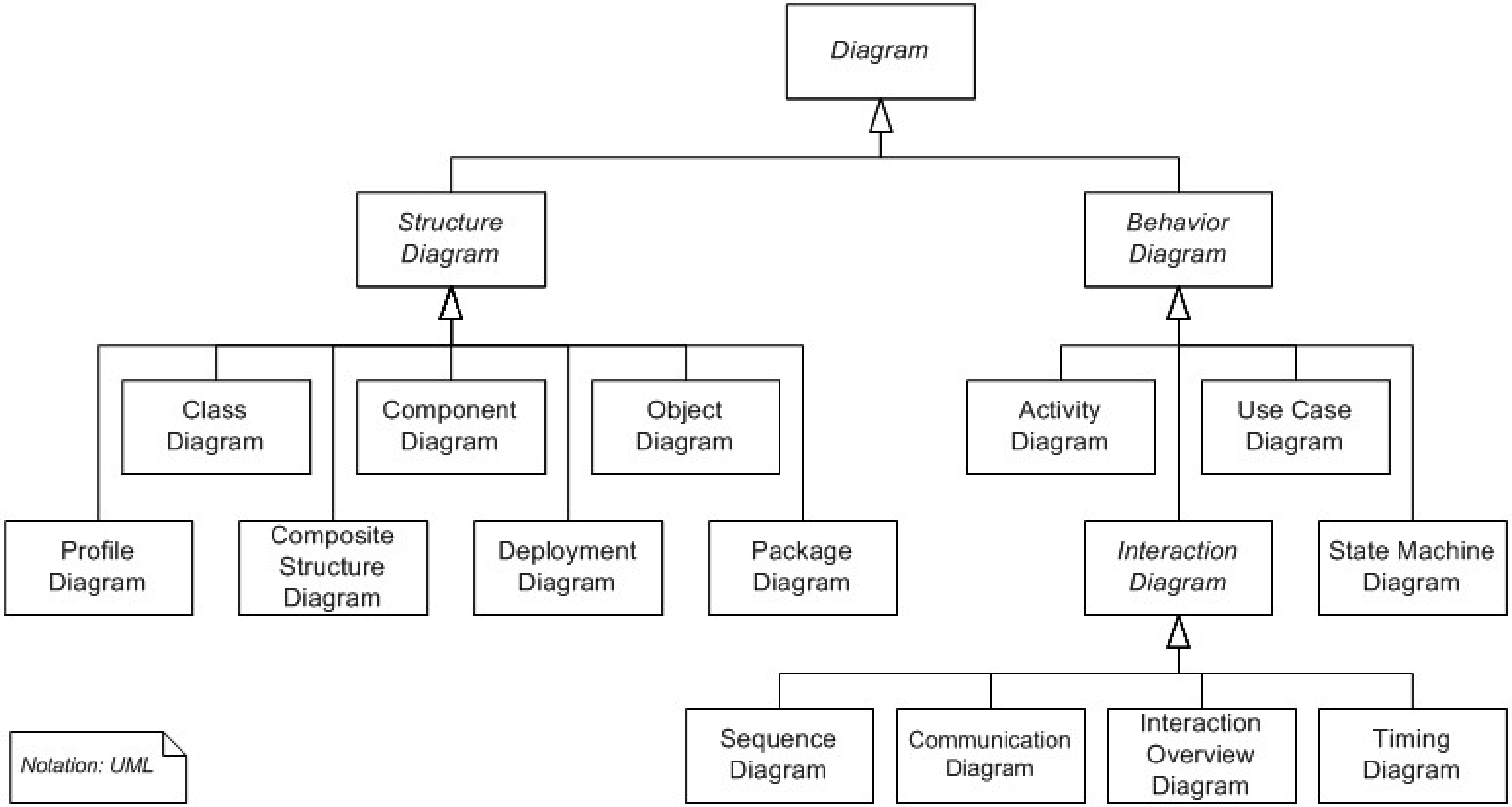
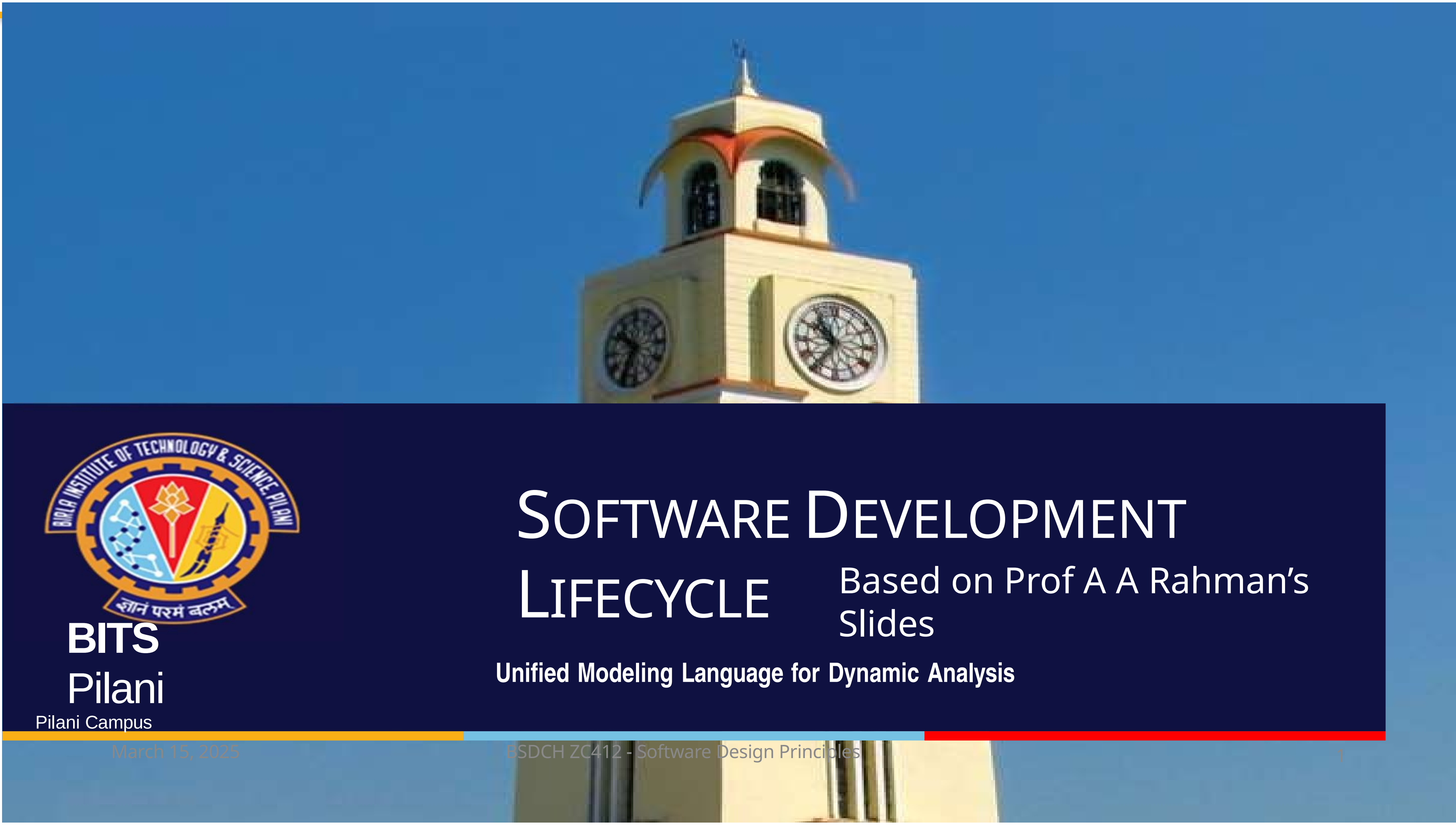


Diagram
Structure Diagram
Behavior Diagram
Class Diagram
Component Diagram
Object Diagram
Profile Diagram
Composite Structure Diagram
Deployment Diagram
Package Diagram
Activity Diagram
Use Case Diagram
Interaction Diagram
State Machine Diagram
Sequence Diagram
Communication Diagram
Interaction Overview Diagram
Timing Diagram
Notation: UML



SOFTWARE DEVELOPMENT LIFECYCLE

Based on Prof A A Rahman's Slides

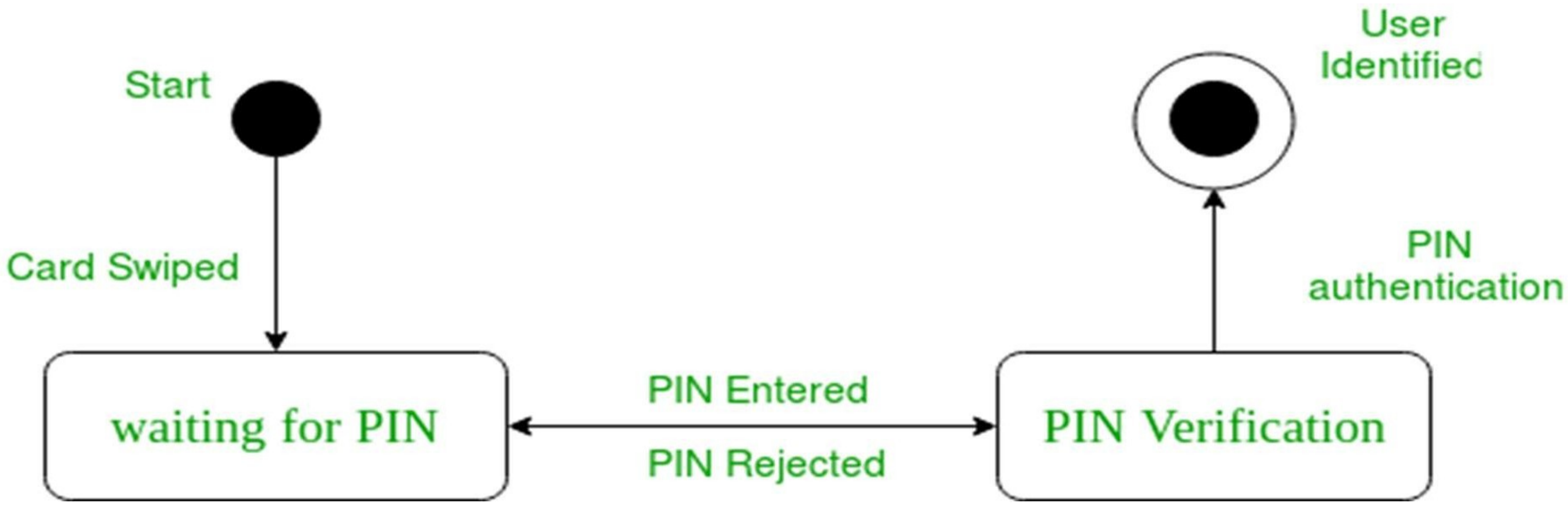
Unified Modeling Language for Dynamic Analysis

BITS
Pilani
Pilani Campus

Unified Modeling Language (UML) State Diagrams (Contd..)

state diagram

- The state diagram shows the different states in which the verification sub-system or class exist for a particular system.


Start
Card Swiped
User Identified
PIN authentication
waiting for PIN
PIN Entered
PIN Rejected
PIN Verification

state diagram

- State Diagrams are used to model and present the dynamic nature of a system.

- State Diagrams consists of different states which represent an activity or an action corresponding to an event

- An event causes the transitions from a state to another state in the state diagram.

- Hence a state diagram is a pictorial representation of the flow of control with respect to either internal or external events.

The State Diagram for Online Banking System is described as below –

- The user can access services by logging in the system using valid login credentials which are unique for every registered customer at the bank.
- As the first state in the diagram represents the action to enter login credentials to verify and validate the user.
- However, if the login credentials are incorrect there occurs a transition to the next state to display the error message.
- Also, the user can now retry to log in the system.
- After several attempts, if the user fails to enter the correct OTP(One Time Password) the user can not try again and the process encounters a final state.

The State Diagram for Online Banking System is described as below –

- If the user credentials are verified the user is allowed to either check balance or get details of a cheque under processing or avail other services offered by the bank.

- If the action is to check balance, the system retrieves the data for the user from the database and the balance in the account of the user is displayed in the next state.

- The process to check balance terminates and the process encounters the final stage.

The State Diagram for Online Banking System is described as below –

- However if the user chooses to initiate a transaction, the user selects to initiate a transaction represented as a specific state in the diagram.

- Also, an after the transaction is successfully initiated the user has to select the account type in the next state.

- The user selects the account, which acts as an internal event to trigger the next state which allows the user to enter the amount to be transacted over.

- After the transaction is executed the flow of control again shifts to check balance where the user can check the balance in the account after the transaction.

- The process again terminates as described above at the final stage.

The State Diagram for Online Banking System is described as below –

- The user can also enlist cheques, approve cheques and get cheque details in favour of the user at the bank which is also represented as actions with directed associations in the diagram.

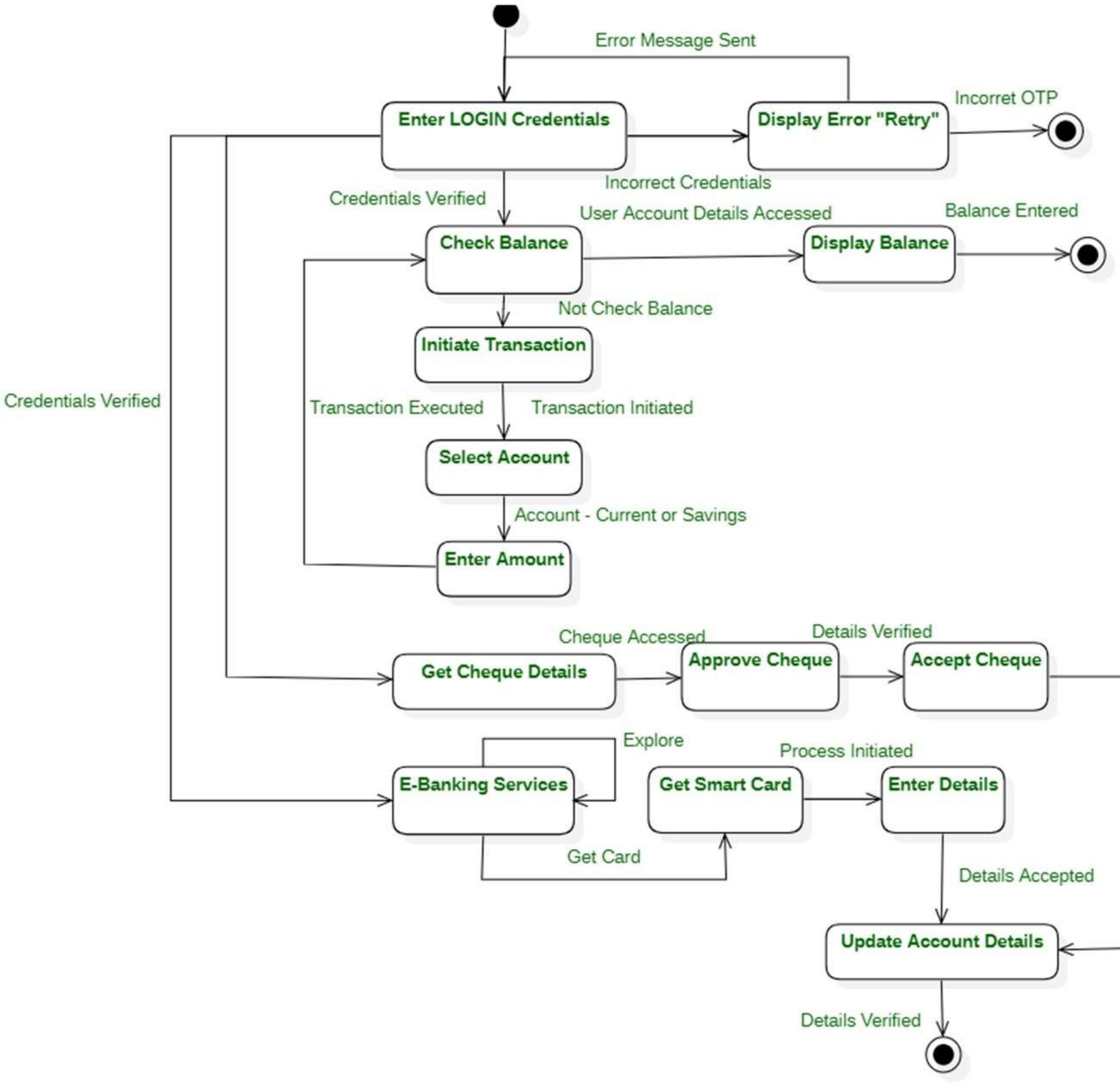
- The user can avail other E-Banking Services where the user can get smart cards.

- The initiation of the process leads to the next state where the user can submit the details to initiate the registration to get a smart card.

- At the end the user can update the account details and exit from the system which is the final state as represented in the diagram.

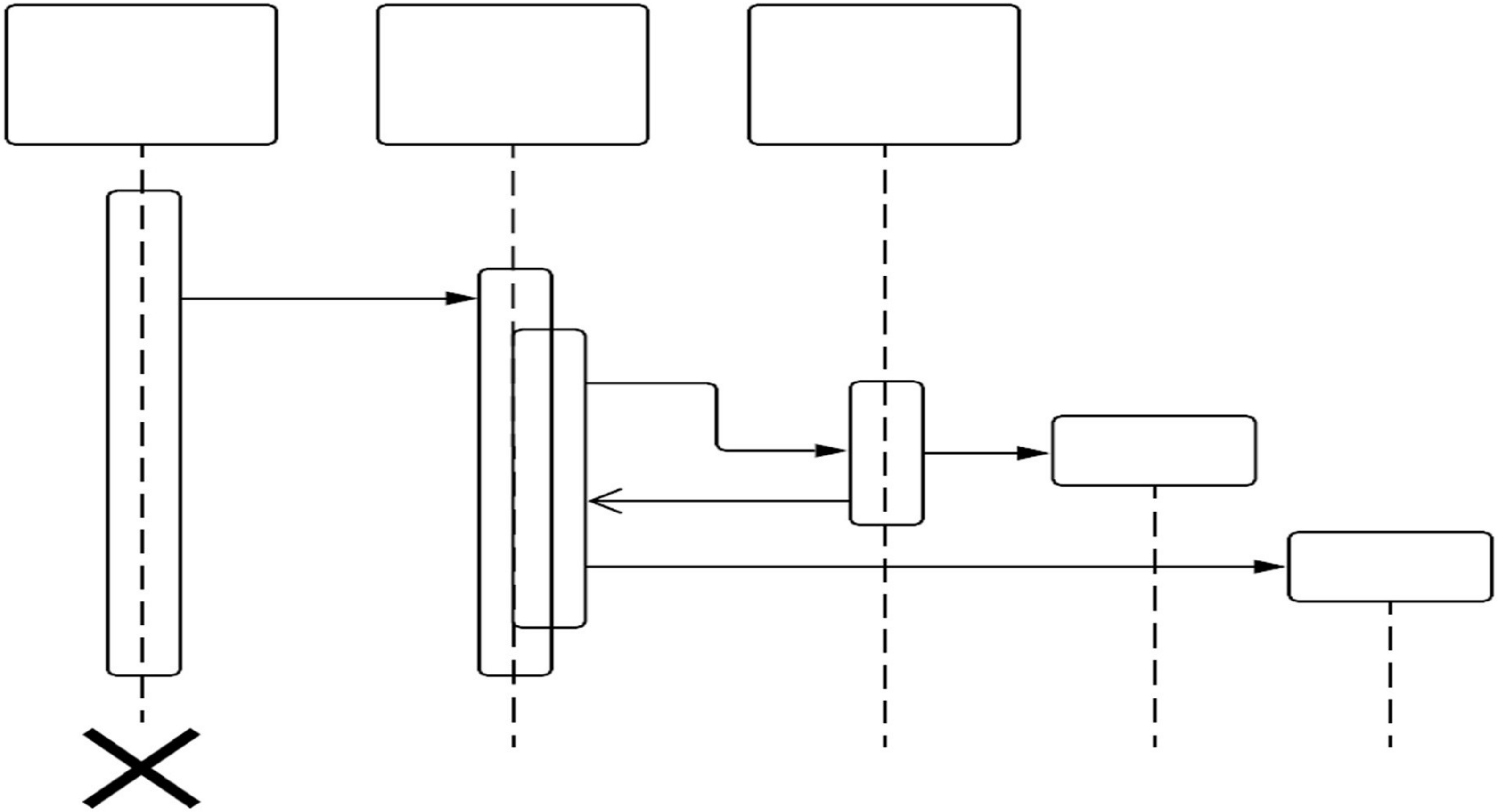
- Hence a State Diagram is able to show the response or behaviour of all the classes in the system to external or internal stimuli.

Online Banking System


Error Message Sent
Enter LOGIN Credentials
Display Error "Retry"
Incorret OTP
Incorrect Credentials
Credentials Verified
User Account Details Accessed
Balance Entered
Check Balance
Display Balance
Not Check Balance
Initiate Transaction
Credentials Verified
Transaction Executed
Transaction Initiated
Select Account
Account - Current or Savings
Enter Amount
Cheque Accessed
Details Verified
Get Cheque Details
Approve Cheque
Accept Cheque
Explore
Process Initiated
E-Banking Services
Get Smart Card
Enter Details
Get Card
Details Accepted
Update Account Details
Details Verified

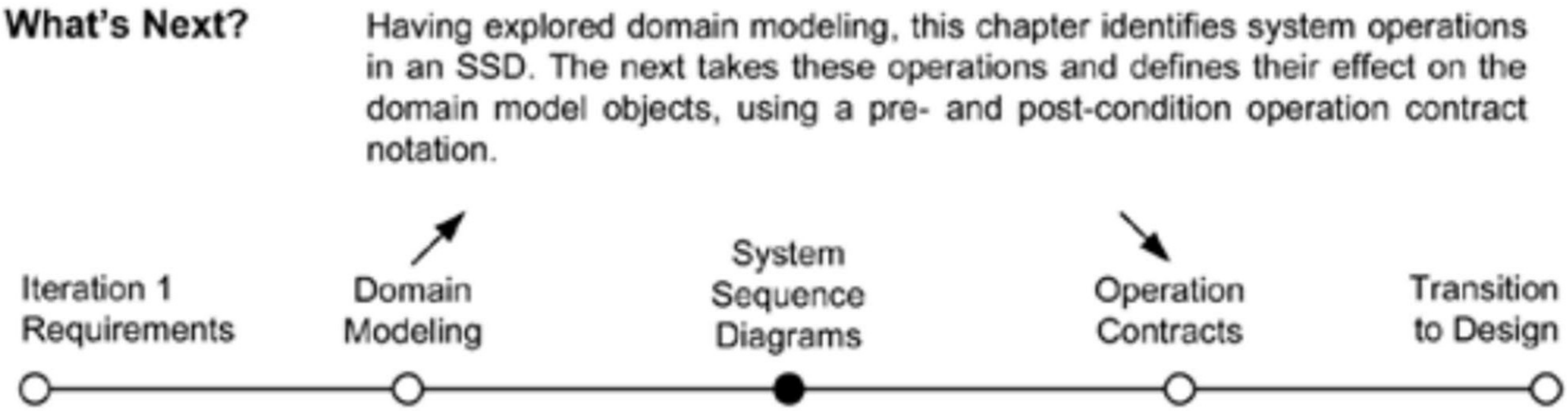
Drawing System Sequence Diagrams

Slides adapted from Applying UML and
Patterns Craig Larman



system sequence diagram

- A system sequence diagram (SSD) is a fast and easily created artifact that illustrates input and output events related to the systems under discussion.
- They are input to operation contracts and most importantly object design.
- The UML contains notation in the form of sequence diagrams to illustrate events from external actors to a system.



What's Next?

Having explored domain modeling, this chapter identifies system operations in an SSD. The next takes these operations and defines their effect on the domain model objects, using a pre- and post-condition operation contract notation.

Iteration 1 Requirements
Domain Modeling
System Sequence Diagrams
Operation Contracts
Transition to Design

system sequence diagram

- The use case text and its implied system events are the inputs to SSD creation.

- The SSD operations (such as enterItem) can in turn be analyzed in the operation contracts, detailed in the Glossary, and most important serve as the starting point for designing collaborating objects.

UP Artifacts

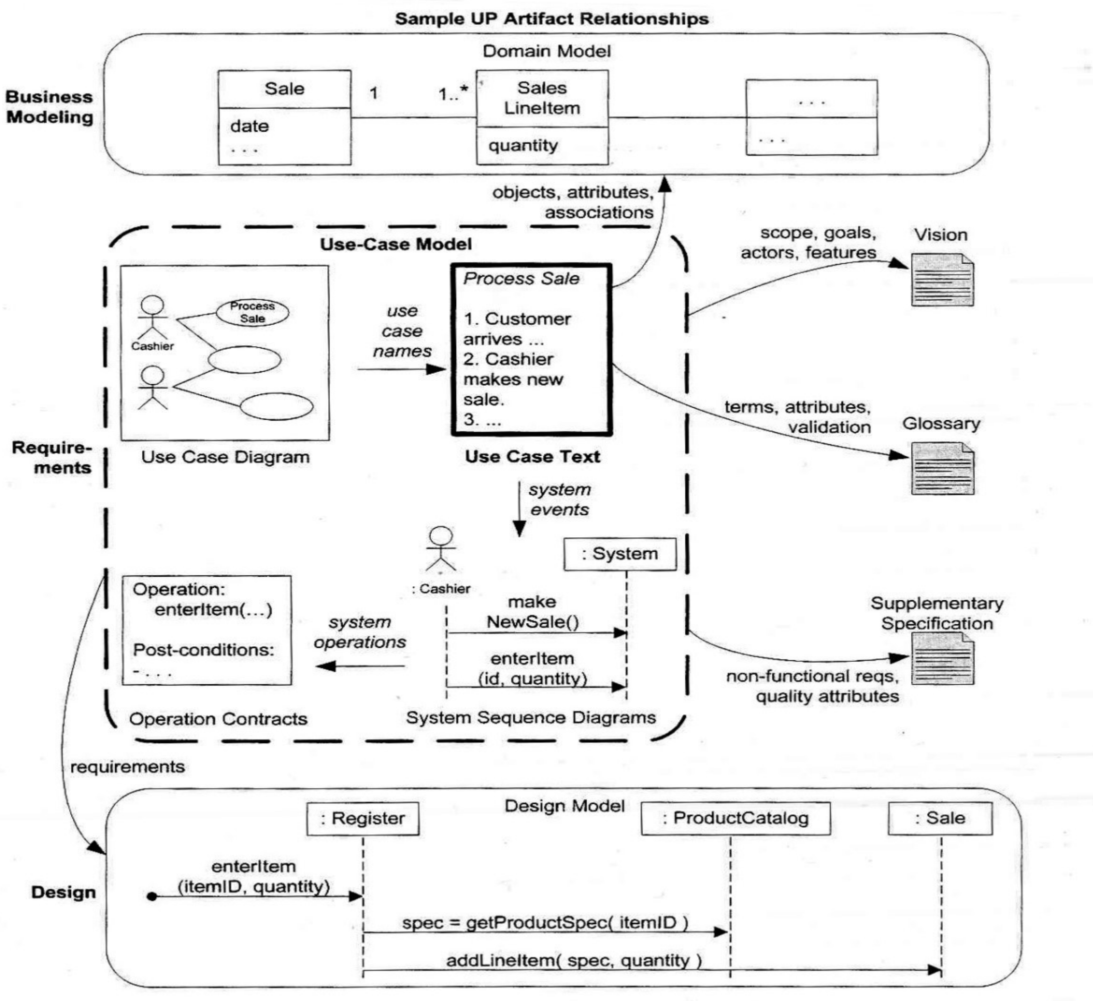
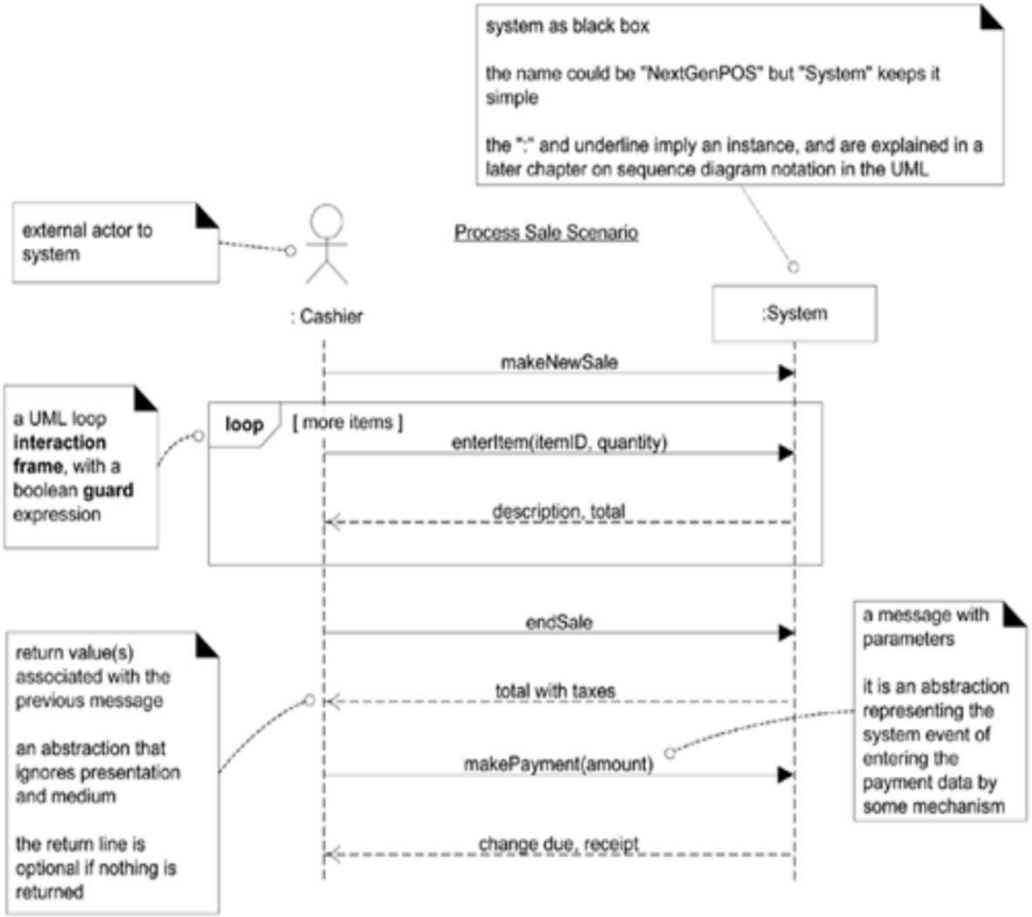

Sample UP Artifact Relationships
Business Modeling
Domain Model
Sale
date
. . .
1
1..*
Sales LineItem
quantity
. . .
. . .
objects, attributes, associations
Use-Case Model
Process Sale
Cashier
Use Case Diagram
use case names
Process Sale
1. Customer arrives ...
2. Cashier makes new sale.
3. ...
Use Case Text
Require-ments
scope, goals, actors, features
Vision
terms, attributes, validation
Glossary
system events
: System
: Cashier
make NewSale()
enterItem (id, quantity)
System Sequence Diagrams
Operation: enterItem(...)
Post-conditions: - . . .
Operation Contracts
system operations
Supplementary Specification
non-functional reqs, quality attributes
requirements
Design
Design Model
: Register
: ProductCatalog
: Sale
enterItem (itemID, quantity)
spec = getProductSpec(itemID)
addLineItem(spec, quantity)

Figure 6.1 Sample UP artifact influence.

- An SSD shows, for a particular course of events within a use case, the external actors that interact directly with the system, the system (as a black box), and the system events that the actors generate

- Time proceeds downward, and the ordering of events should follow their order in the scenario.

- The main success scenario of a cash-only Process Sale scenario. It indicates that the cashier generates makeNewSale, enterItem, endSale, and makePayment system events. These events are implied or suggested by reading through the use case text.


system as black box
the name could be "NextGenPOS" but "System" keeps it simple
the ":" and underline imply an instance, and are explained in a later chapter on sequence diagram notation in the UML
external actor to system
Process Sale Scenario
: Cashier
:System
makeNewSale
a UML loop interaction frame, with a boolean guard expression
loop
[more items]
enterItem(itemID, quantity)
description, total
endSale
a message with parameters
it is an abstraction representing the system event of entering the payment data by some mechanism
return value(s) associated with the previous message
an abstraction that ignores presentation and medium
the return line is optional if nothing is returned
total with taxes
makePayment(amount)
change due, receipt

Why use System Sequence Diagram?

- The **System Sequence Diagram**, aka **SSD**, is the sub-type of the sequence diagram that shows us the input and output events.
- The system sequence diagram illustrates the sequence of use cases.
- These use cases are the events caused by external actors.
- It defines their order and the events occurring inside the system.

Why use System Sequence Diagram?

- The System Sequence Diagram (SSD) tracks how the functions and the use case functions are performed inside the system.
- We use these diagrams to model our software in concern with how the system interacts with the events.
- This diagram is used to create a blueprint derived from the main success scenario, and then we further design by seeing other cases and scenarios.

What are System Sequence Diagrams?

- Use cases describe how external actors interact with the software system we are interested in creating.

- During this interaction an actor generates system events to a system, usually requesting some system operation to handle the event.

- For example, when a cashier enters an item's ID, the cashier is requesting the POS system to record that item's sale (the enterItem event).

What are System Sequence Diagrams?

- That event initiates an operation upon the system.
- The use case text implies the enterItem event, and the SSD makes it concrete and explicit.
- The UML includes sequence diagrams as a notation that can illustrate actor interactions and the operations initiated by them.

system sequence diagram

- A system sequence diagram is a picture that shows, for one particular scenario of a use case, the events that external actors generate, their order, and inter-system events.

- All systems are treated as a black box; the emphasis of the diagram is events that cross the system boundary from actors to systems.

- Guideline

- Draw an SSD for a main success scenario of each use case, and frequent or complex alternative scenarios

Why Draw an SSD?

- An interesting and useful question in software design is this:

- What events are coming in to our system? Why?

- Because we have to design the software to handle these events (from the mouse, keyboard, another system, ...) and execute a response.

Why Draw an SSD?

- Basically, a software system reacts to three things:
- 1) external events from actors (humans or computers),
- 2) timer events, and
- 3) faults or exceptions (which are often from external sources).
- Therefore, it is useful to know what, precisely, are the external input events the system events.
- They are an important part of analyzing system behavior.

Why Draw an SSD?

- You may be familiar with the idea of identifying the messages that go into one software object.

- But this concept is useful at higher levels of components, including the entire system viewed (abstractly) as one thing or object.

- Before proceeding to a detailed design of how a software application will work, it is useful to investigate and define its behavior as a "black box."

Why Draw an SSD?

- System behavior is a description of what a system does, without explaining how it does it.
- One part of that description is a system sequence diagram.
- Other parts include the use cases and system operation contracts.

Sequence Diagrams

- The UML does not define something called a "system" sequence diagram but simply a "sequence diagram."

- The qualification is used to emphasize its application to systems as black boxes.

- Later, sequence diagrams will be used in another context to illustrate the design of interacting software objects to fulfill work.

Sequence Diagrams

- Shows objects and classes involved in a use case scenario

- Shows the messages exchanged between objects in time order sequence

- Used in design to assign object responsibilities

- Can be used test user interface requirements

System Sequence Diagrams (SSDs)

- SSD is an artifact of analysis that illustrates input and output events related to the system.

- SSD is associated with use-case realization in the logical view of system development.

What are Elements of an SSD?

- Some of the elements that are used in the use case diagram are also as follows here.
- The essential elements of the system sequence diagram are.
- Objects
- Actors
- Events

- **Objects**
- In a traditional UML, the box depicting a title represents the class or an object but, in System Sequence Diagram, this shape represents the system as the black-box.

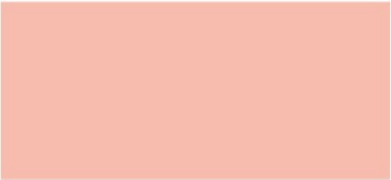
- The black box is a system that is not illuminated to the outside world because they contain inner processes.

- **Actors**
- These are the external entities that interact with our system, and the system is created by keeping actors in mind.

- The interaction can be like interacting with the cashier, customer retrieving amount from the ATM or anything. They are the stick-like figures in the diagram.

Events

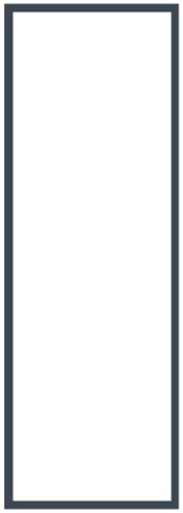
- The interaction or communication of an actor or system to the system is an event.
- Then with the specific event, the system performs the specific task.
- These events are generated in a sequence or a specific order.
- The dashed lines aka, the lifelines, are the events in the system sequence diagram.
- These lines start from the rectangle shape o with an actor.



Object
symbol

- Represents a class or object in UML.

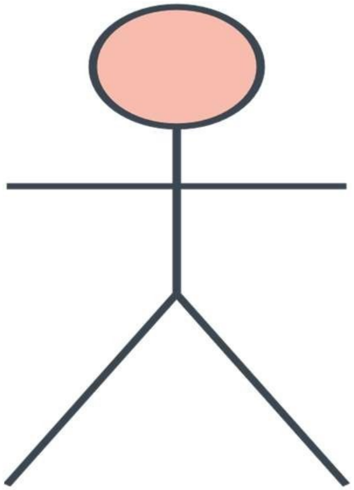
- The object symbol demonstrates how an object will behave in the context of the system.
- Class attributes should not be listed in this shape.



Activation box

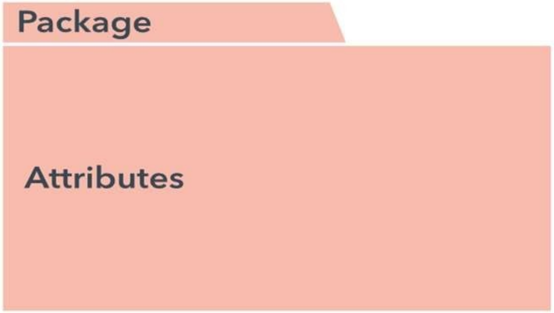
Represents the time needed for an object to complete a task.

The longer the task will take, the longer the activation box becomes.



Actor
symbol

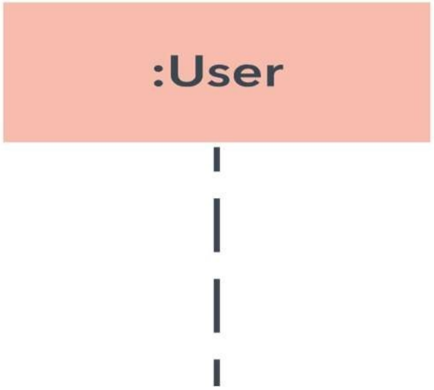
Shows entities that interact with or are external to the system.



Package symbol

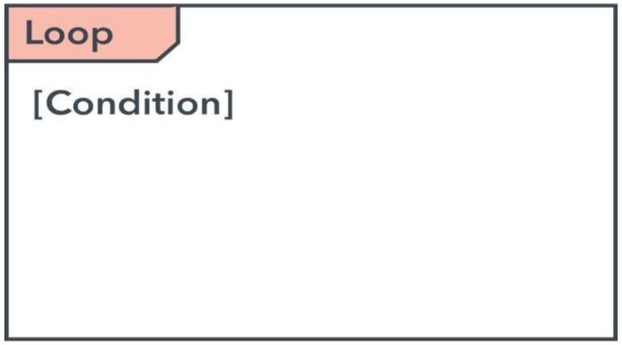
- Used in UML 2.0 notation to contain interactive elements of the diagram.

- Also known as a frame, this rectangular shape has a small inner rectangle for labeling the diagram.



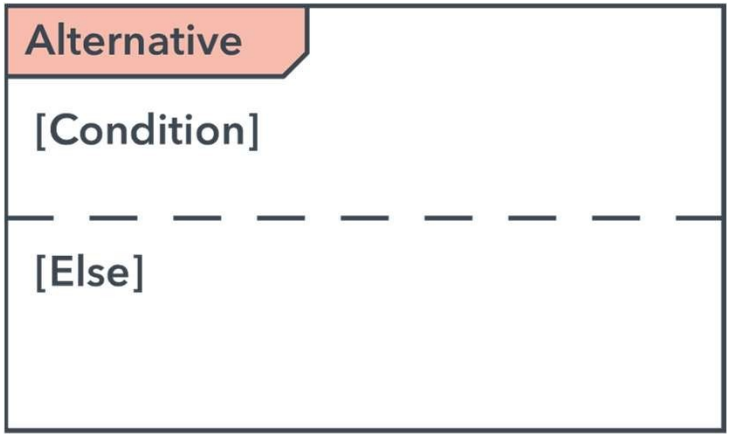
Lifeline
symbol

- Represents the passage of time as it extends downward.
- This dashed vertical line shows the sequential events that occur to an object during the charted process.
- Lifelines may begin with a labeled rectangle shape or an actor symbol.



Option loop symbol

- Used to model if/then scenarios, i.e., a circumstance that will only occur under certain conditions.



Alternative symbol

- Symbolizes a choice (that is usually mutually exclusive) between two or more message sequences.
- To represent alternatives, use the labeled rectangle shape with a dashed line inside.

Use cases for sequence diagrams

- The following scenarios are ideal for using a sequence diagram:
- **Usage scenario:** A usage scenario is a diagram of how our system could potentially be used.
- It's a great way to make sure that we have worked through the logic of every usage scenario for the system.
- **Method logic:** Just as we might use a UML sequence diagram to explore the logic of a use case, we can use it to explore the logic of any function, procedure, or complex process.
- **Service logic:** If we consider a service to be a high-level method used by different clients, a sequence diagram is an ideal way to map that out.

SSDs vs. Sequence Diagrams

Sequence Diagram

In the Unified Modeling Language (UML), the sequence diagrams depict the relationship between the actors and objects in a system with respect to time.

The sequence diagram shows the control flow of these relations and how the system works regarding these relationships of objects and the actors.

When the objects become so complex in a system design, this diagram is made to model the control flow and relationship between actors and objects.

System Sequence Diagram

The System Sequence Diagram assumes the higher abstraction level by visualizing the control flow of the system and the actor.

Visualizing the interaction between the classes and objects using a sequence diagram is not part of this analysis.

SSDs and System Behavior

- System behaves as “Black Box”.
- Interior objects are not shown, as they would be on a Sequence Diagram.

:System

Use Cases are Source for SSD

- Use cases describe
  - How actors interact with system.
  - Typical course of events that external actors generate and
  - The order of the events.

SSD Components

For a particular scenario of use-case an SSD shows-

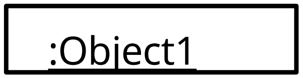
- The external actors that interact directly with the system.
- The System (as a black box).
- The system events that the actors generate.

What SSDs Show

- Match operations of the system in response to the events generated
- Depict the temporal order of the events.
- Should be done for the main success scenario of the use-case
    - Also for frequent and alternative scenarios

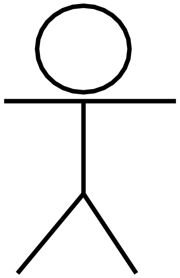
Objects on SSDs

- Objects are instances of classes.
- Represented as a rectangle which contains the name of the object underlined
- Because the system is instantiated, it is shown as an object.



Actors on SSDs

- **Actor:** An Actor is modeled using the ubiquitous symbol, the stick figure.



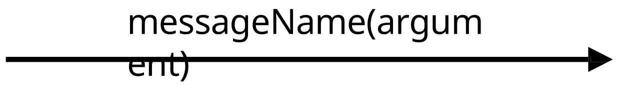
actor
1

Lifelines on SSDs

- LifeLine identifies the existence of the object over time.
- The notation for a Lifeline is a vertical dotted line extending from an object.

Messages on SSDs

- Messages, modeled as horizontal arrows between Activations, indicate the communications between objects.



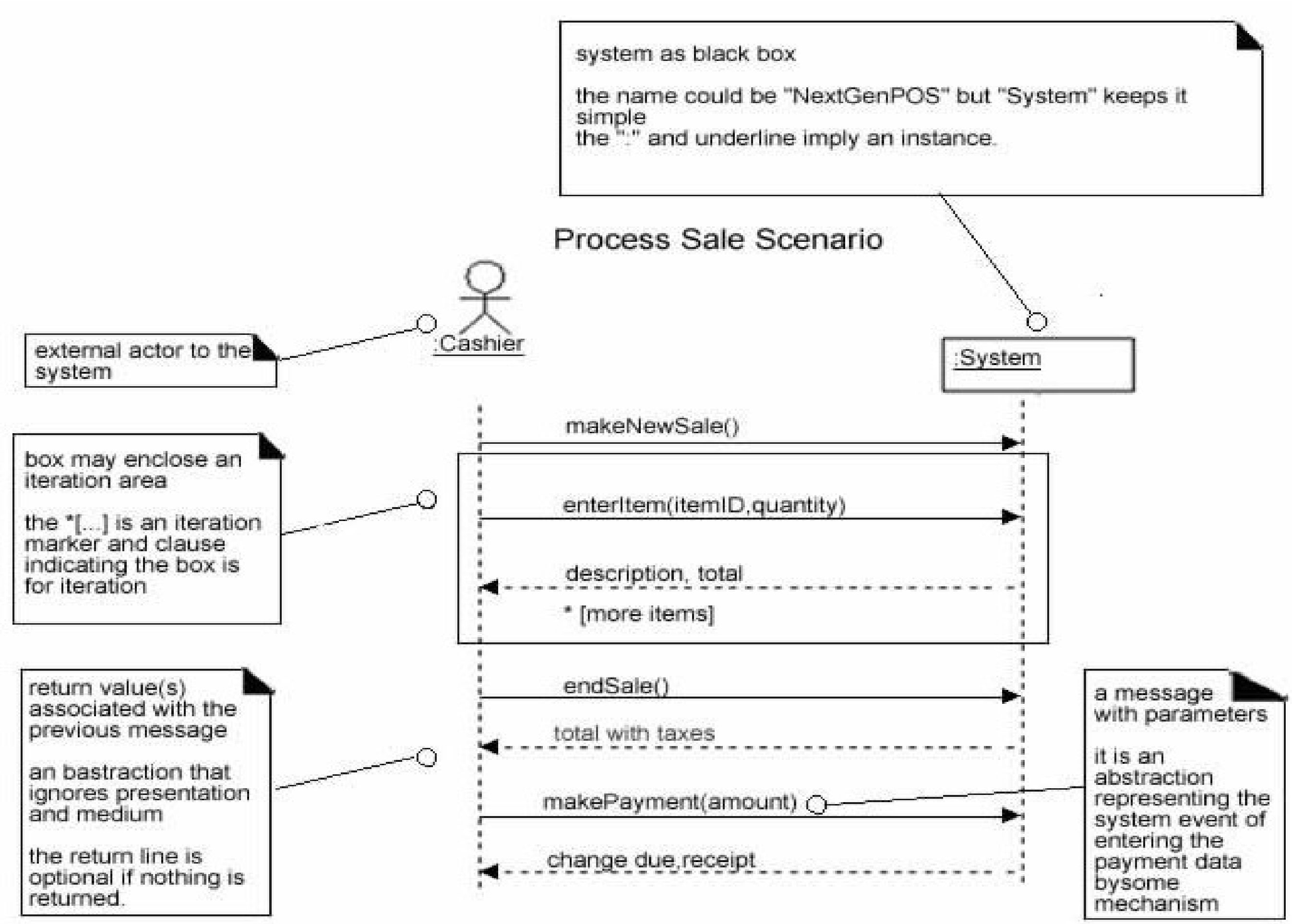
Example of an SSD

- Following example shows the success scenario of the Process Sale use case.

- Events generated by cashier

  (actor)- makeNewSale
  enterItem
  endSale
  makePayment.

SSD for Process Sale scenario



Process Sale Scenario
:Cashier
:System
makeNewSale()
enterItem(itemID,quantity)
description, total
* [more items]
endSale()
total with taxes
makePayment(amount)
change due,receipt

system as black box

the name could be "NextGenPOS" but "System" keeps it simple
the ":" and underline imply an instance.

external actor to the system

box may enclose an iteration area

the *[...] is an iteration marker and clause indicating the box is for iteration

return value(s) associated with the previous message

an bastraction that ignores presentation and medium

the return line is optional if nothing is returned.

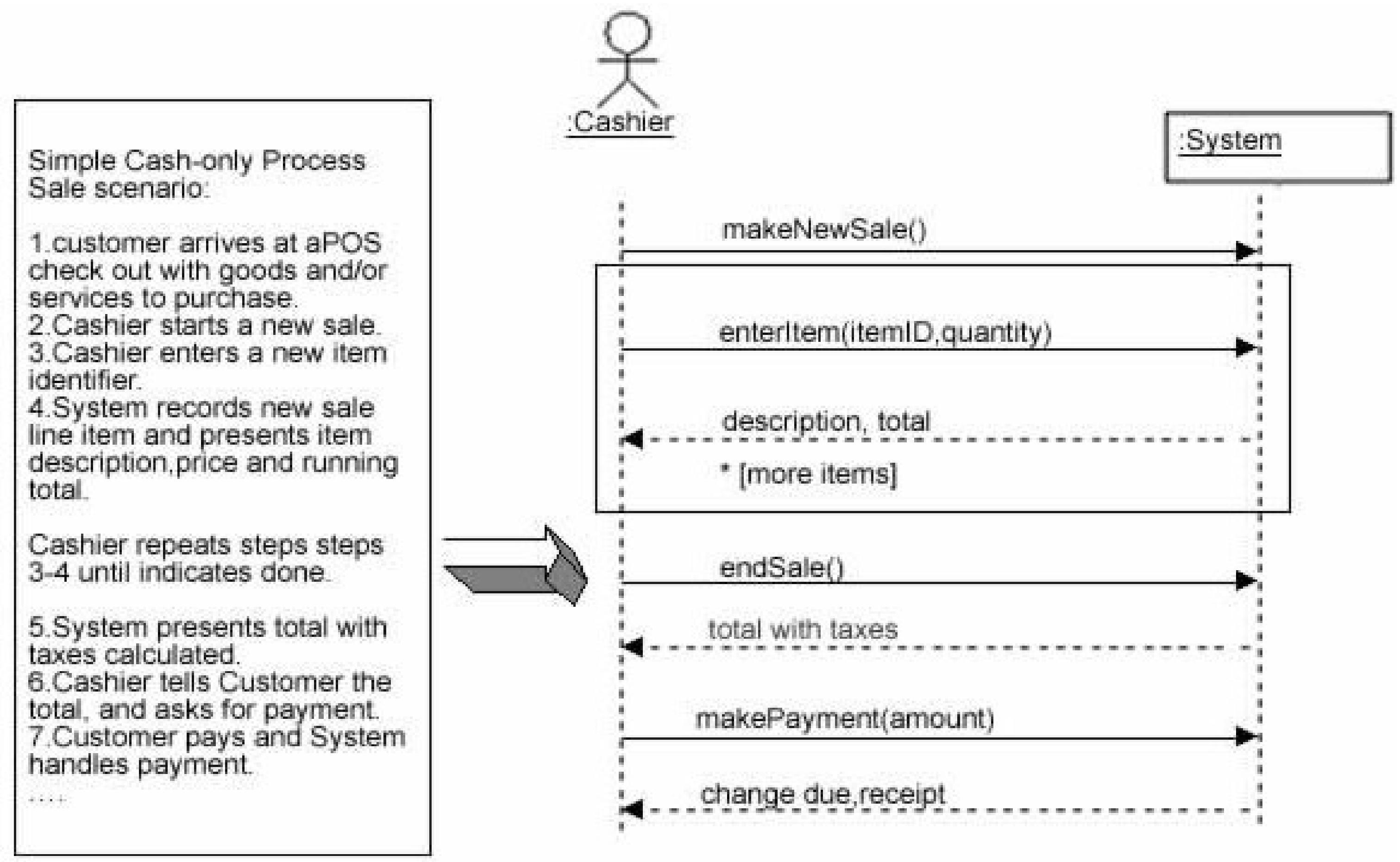
a message with parameters

it is an abstraction representing the system event of entering the payment data bysome mechanism

Create SSDs for each Use Case

1. Draw a lifeline representing the system as a black box.
2. Identify *each* actor that directly operates on the system.
3. Draw a lifeline for *each* actor
4. From the use case happy path text, identify system (external) events that actors generate (look at right side of the flow of events).
5. Add them as messages to diagram.
6. Add the main outputs from the use case as messages back to actor – see use case table
7. Optionally, include the use case text to the left of the diagram.

SSDs are derived from use cases.



Simple Cash-only Process Sale scenario:

1.customer arrives at aPOS check out with goods and/or services to purchase.
2.Cashier starts a new sale.
3.Cashier enters a new item identifier.
4.System records new sale line item and presents item description,price and running total.

Cashier repeats steps steps 3-4 until indicates done.

5.System presents total with taxes calculated.
6.Cashier tells Customer the total, and asks for payment.
7.Customer pays and System handles payment.
....

:Cashier
:System
makeNewSale()
enterItem(itemID,quantity)
description, total
* [more items]
endSale()
total with taxes
makePayment(amount)
change due,receipt

System Events and System Boundary

- To identify the system events, knowing the system boundary is critical.

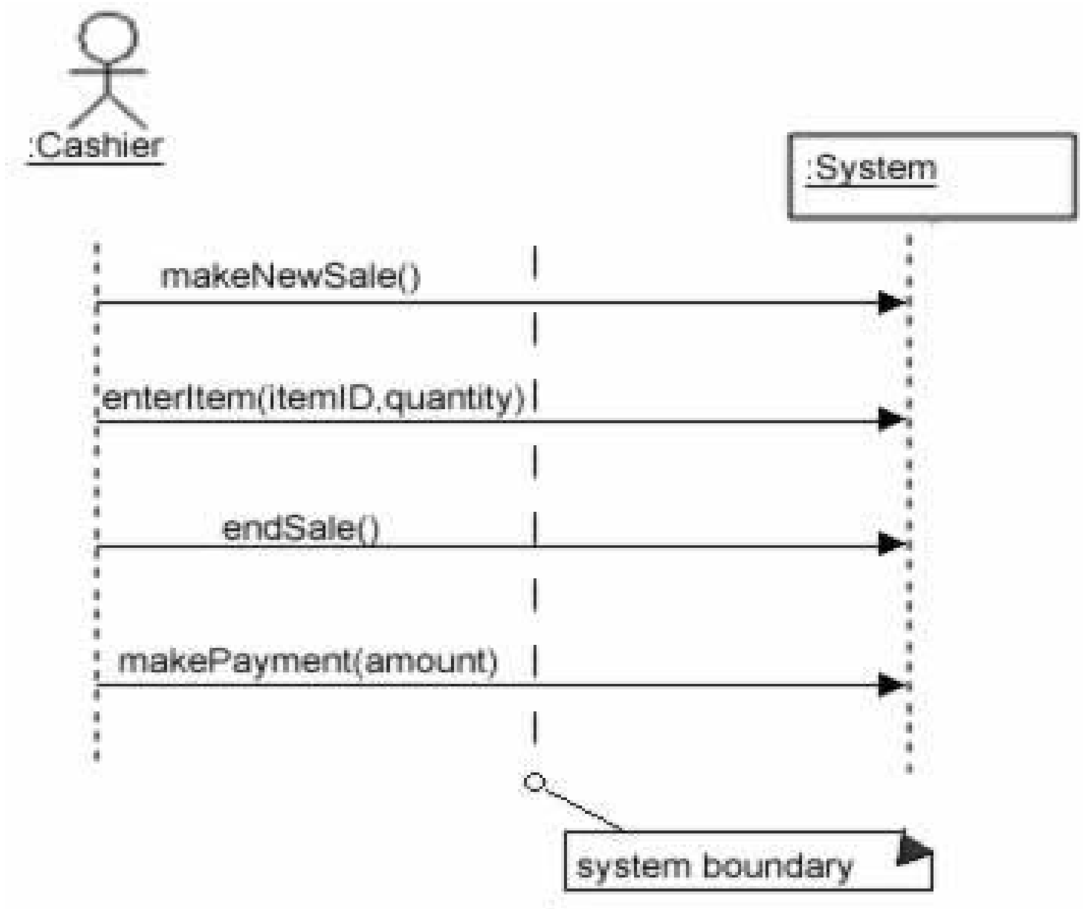
- For the purpose of software development, the system boundary is chosen to be the software system itself.

Determining SSD System Boundary

- Identifying the System events-

1. Determine the actors that directly interact with the system.
2. In the process Sale example, the customer does not directly interact with the POS system.
3. Cashier interacts with the system directly.
4. Therefore cashier is the generator of the system events.

Defining system boundary.


:Cashier
:System
makeNewSale()
enterItem(itemID,quantity)
endSale()
makePayment(amount)
system boundary

Naming System Events and Operations

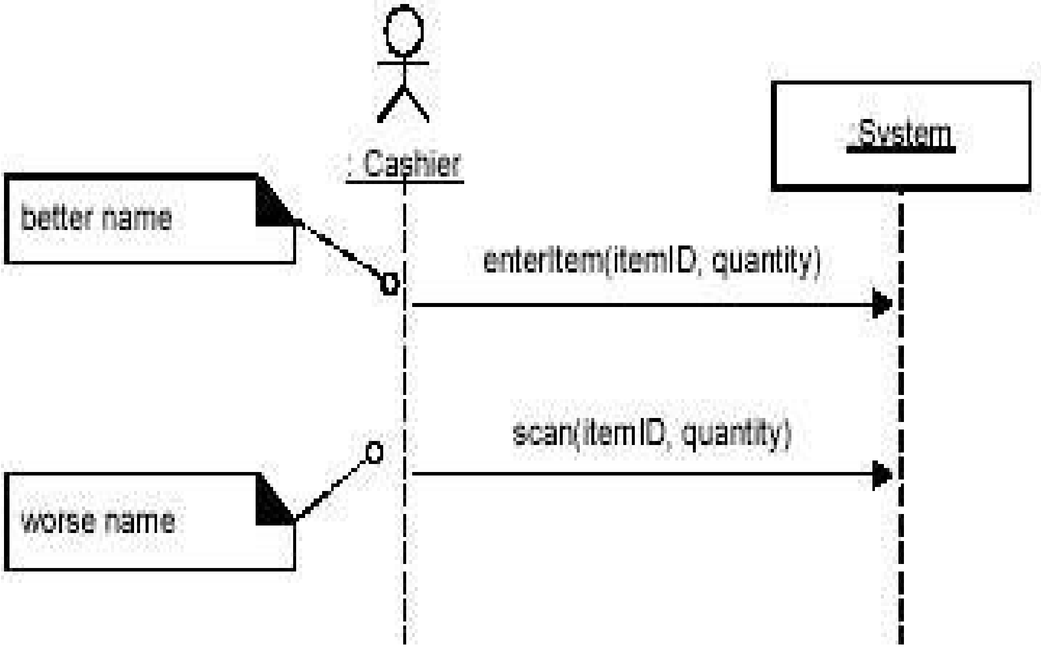
System event

- External input event generated by an actor.
- Initiates a responding operation by system.
- In order to improve the clarity, it is appropriate to start the name of the system event with a verb

System operation

- Operation invoked in response to system event.

Choosing SSD event / operation names


: Cashier
:System
better name
worse name
enterItem(itemID, quantity)
scan(itemID, quantity)

“enterItem” is better than “scan” as it captures the intent of operation rather than what interface is used to capture the system event (design choice).

SSDs in Analysis

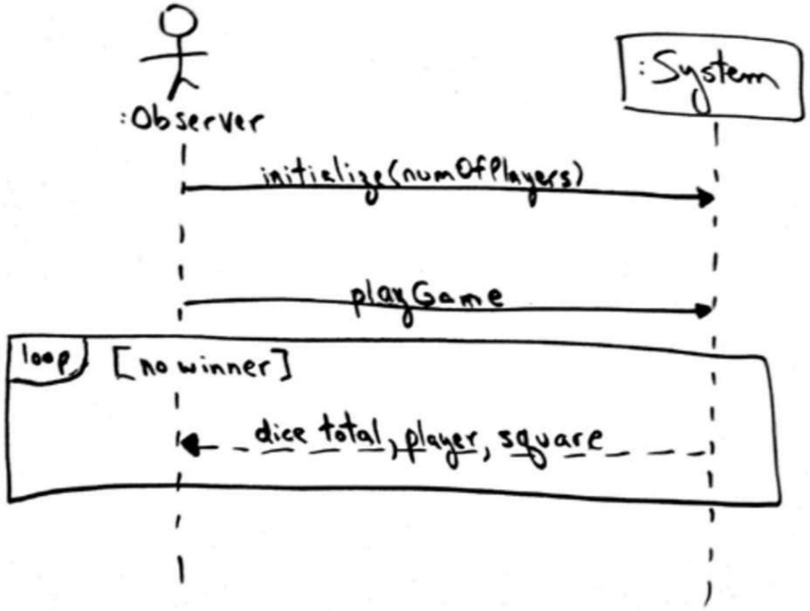
SSDs are a visualization of the interactions implied in the use cases.

It is useful to create SSDs during analysis to:

- Identify the system events and major operations
- Write system operation contracts (Contracts describe detailed system behavior)
- Provide a way for us to visually step through invocation of the operations in Use-Cases.

SSD for a Play Monopoly Game scenario.

- Monopoly SSD The Play Monopoly Game use case is simple, as is the main scenario.
- The observing person initializes with the number of players, and then requests the simulation of play, watching a trace of the output until there is a winner.


:Observer
:System
initialize(numOfPlayers)
playGame
loop
[no winner]
dice total, player, square

Iterative and Evolutionary SSDs

- Don't create SSDs for all scenarios.
- Unless you are using an estimation technique (such as function point counting) that requires identification of all system operations.
- Draw them only for the scenarios chosen for the next iteration.
- They shouldn't take long to sketch perhaps a few minutes or a half hour.
- SSDs are also very useful when you want to understand the interface and collaborations of existing systems, or to document the architecture.

SSDs Within the UP

- SSDs are part of the Use-Case Modela visualization of the interactions implied in the scenarios of use cases.

- SSDs were not explicitly mentioned in the original UP description, although the UP creators are aware of and understand the usefulness of such diagrams.

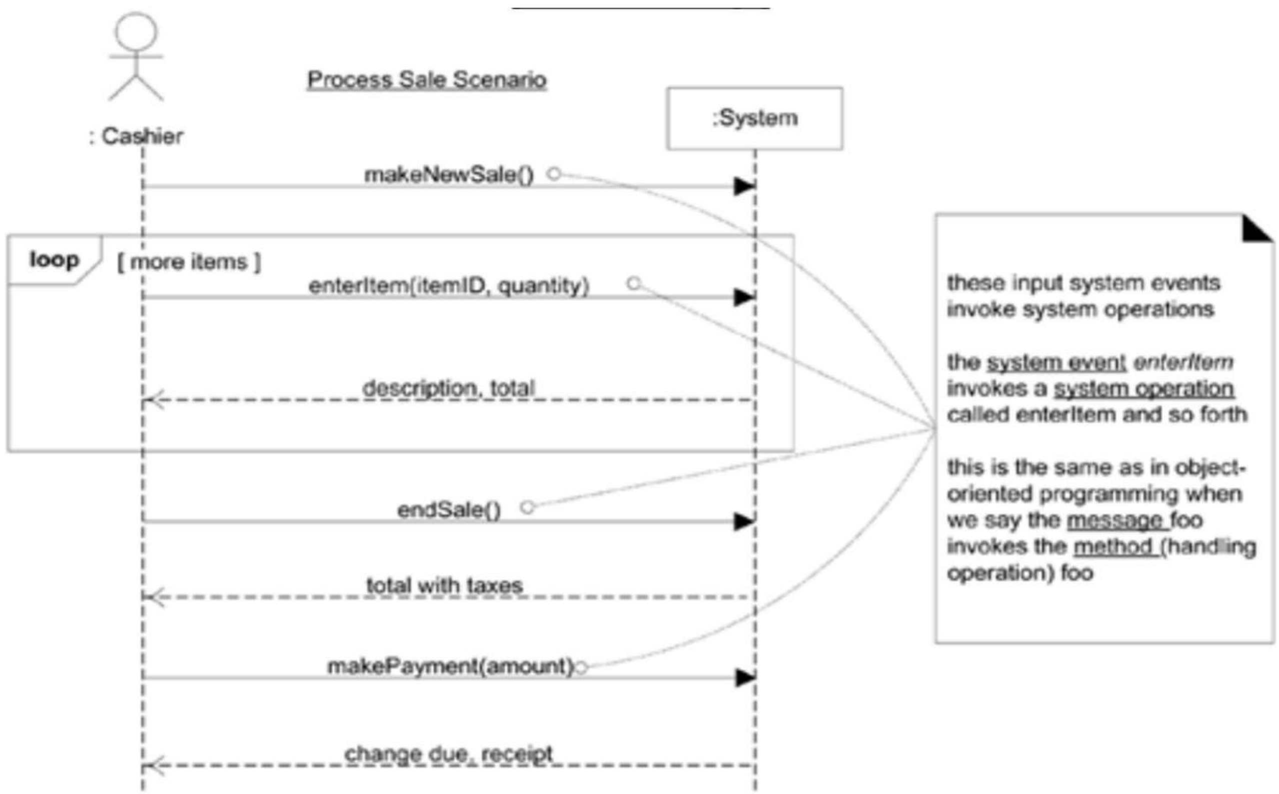
- SSDs are an example of the many possible skillful and widely used analysis and design artifacts or activities that the UP or RUP documents - do not mention.

- But the UP, being very flexible, encourages the inclusion of any and all artifacts and practices that add value.

What is a System Operation?

- Operation contracts may be defined for system operations
- operations that the system as a black box component offers in its public interface.
- System operations can be identified while sketching SSDs
- To be more precise, the SSDs show system events events or I/O messages relative to the system.
- Input system events imply the system has system operations to handle the events, just as an OO message (a kind of event or signal) is handled by an OO method (a kind of operation).

SSD. System operations handle input system events.


Process Sale Scenario
: Cashier
:System
makeNewSale()
loop
[more items]
enterItem(itemID, quantity)
description, total
endSale()
total with taxes
makePayment(amount)
change due, receipt

these input system events invoke system operations

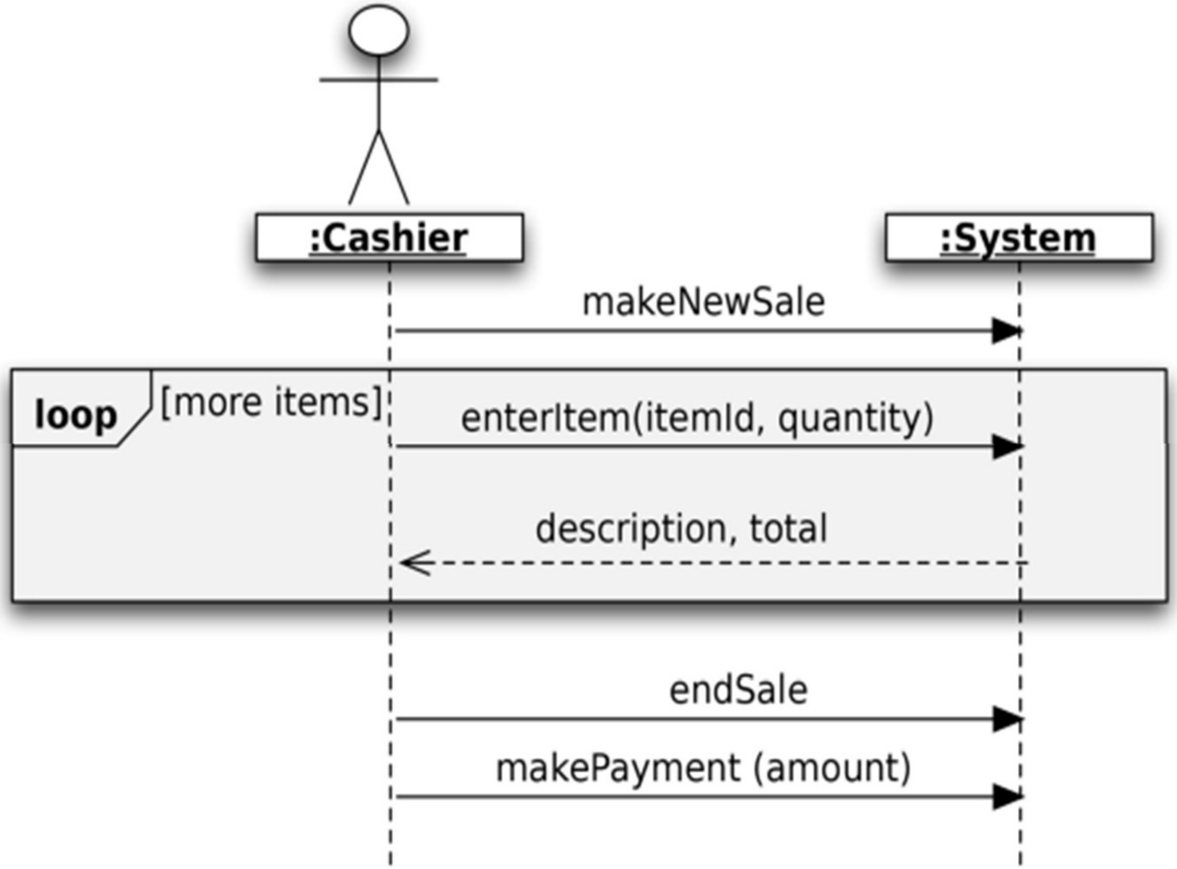
the system event *enterItem* invokes a system operation called enterItem and so forth

this is the same as in object-oriented programming when we say the message foo invokes the method (handling operation) foo

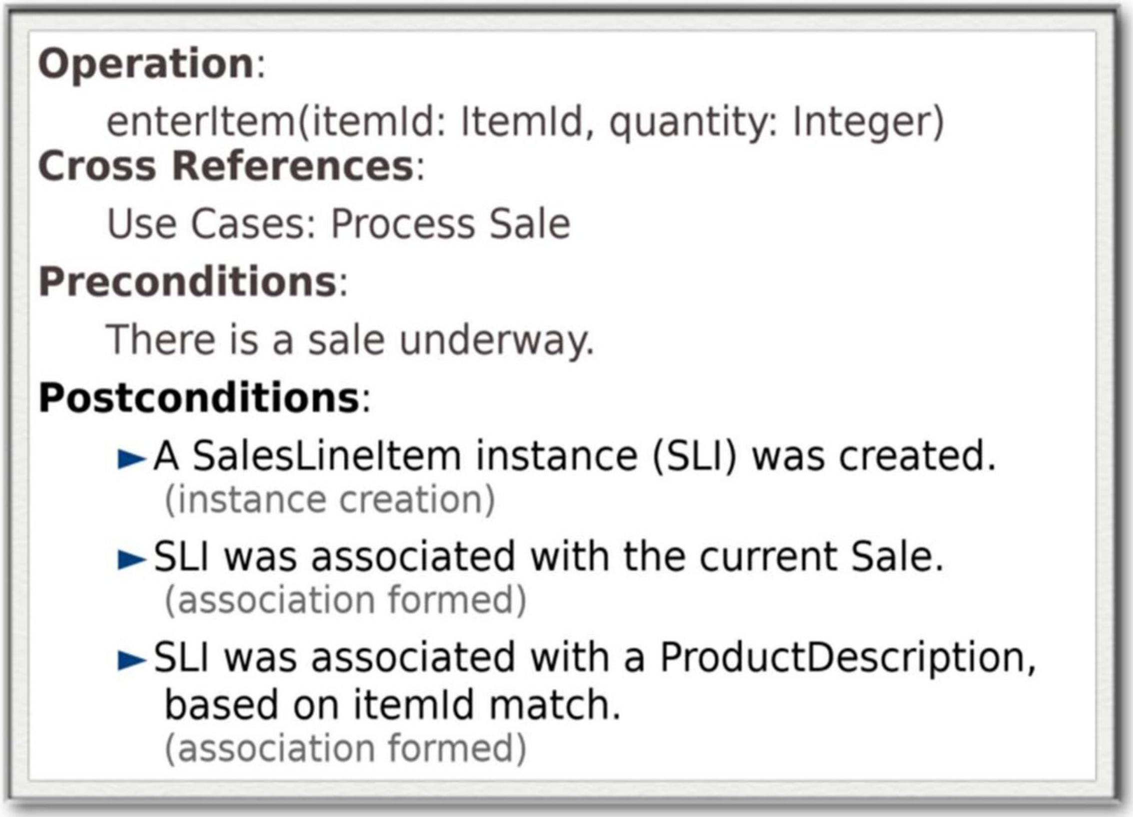
The entire set of system operations, across all use cases, defines the public system interface, viewing the system as a single component or class. In the UML, the system as a whole can be represented as one object of a class named (for example) System

The system operations are shown in the system sequence diagram (SSD)

To provide more analysis detail on the effect of the system operations implied use cases, (System) operation Contracts may be considered


:Cashier
:System
makeNewSale
loop
[more items]
enterItem(itemId, quantity)
description, total
endSale
makePayment (amount)

Operation Contract for “enterItem()”



Operation:

enterItem(itemId: ItemId, quantity: Integer)

Cross References:

Use Cases: Process Sale

Preconditions:

There is a sale underway.

Postconditions:

- ► A SalesLineItem instance (SLI) was created.
  (instance creation)
- ► SLI was associated with the current Sale.
  (association formed)
- ► SLI was associated with a ProductDescription, based on itemId match.
  (association formed)

Example of an SSD for the Process Sale Scenario

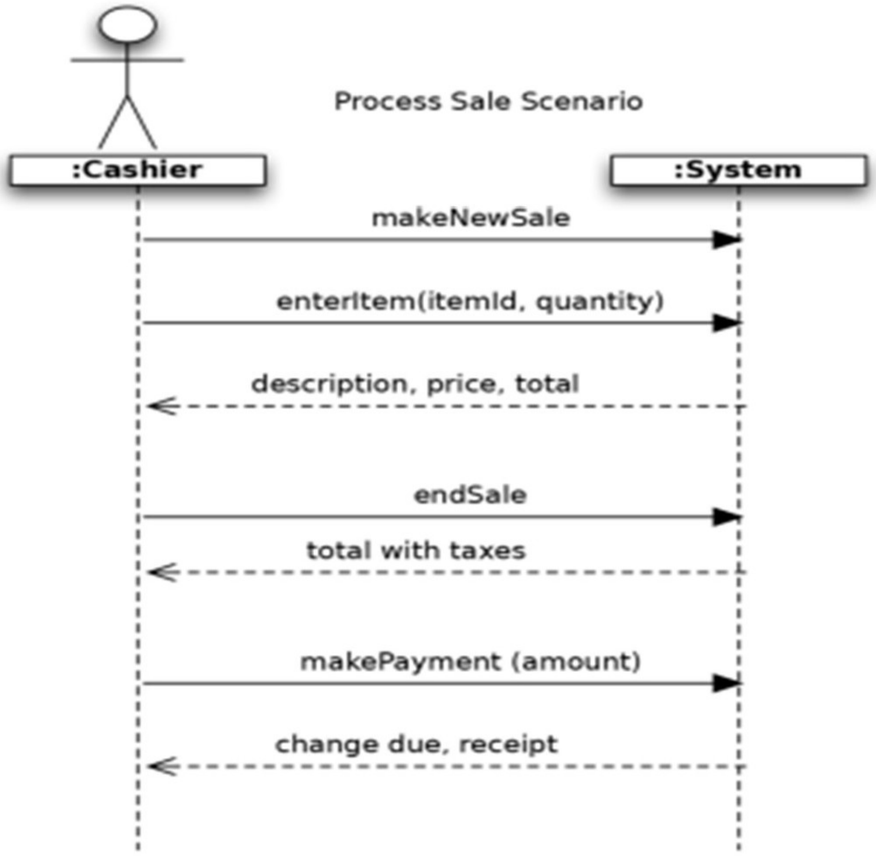
Use Case: Process Sale Scenario – Main Success Story

1. Cashier starts new sale
2. Cashier enters item identifier
3. System records sale line item and presents item description, price and running total – steps 2 and 3 are repeated until all items are processed
4. System presents total with taxes calculated
5. Cashier tells Customer the total and asks for payment
6. Customer pays and System handles payment

Example of an SSD for the Process Sale Scenario

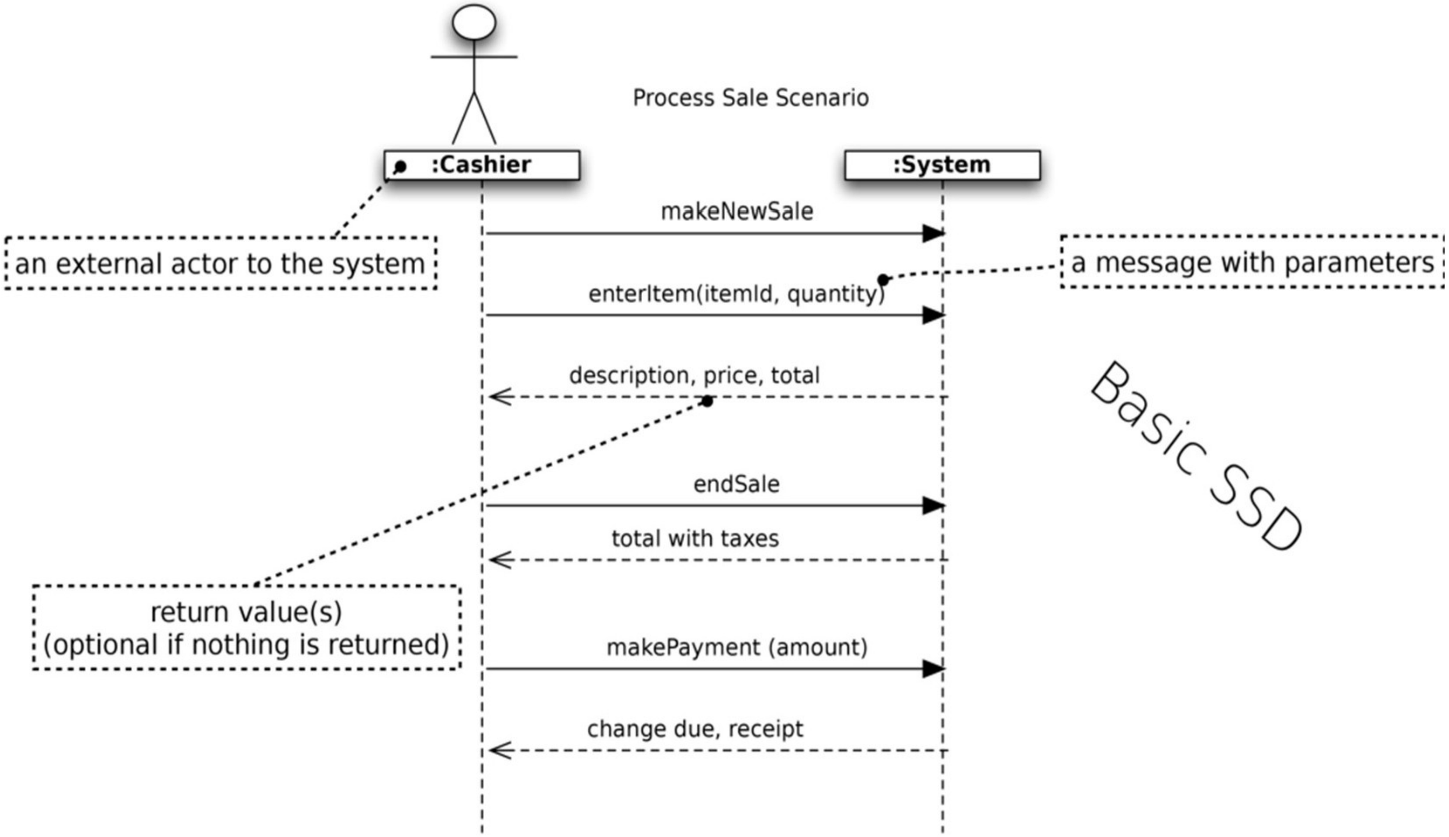
- SSDs are drawn using UML's sequence diagram notation.

- The name of each event should state the intention(e.g. "enterItem(ItemId)" vs "scan (ItemId)").


Process Sale Scenario
:Cashier
:System
makeNewSale
enterItem(itemId, quantity)
description, price, total
endSale
total with taxes
makePayment (amount)
change due, receipt

Example of an SSD for the Process Sale Scenario

- SSDs are drawn using UML's sequence diagram notation. The name of each event should state the intention(e.g. “enterItem(ItemId)” vs “scan (ItemId)”).


Process Sale Scenario
:Cashier
:System
an external actor to the system
makeNewSale
a message with parameters
enterItem(itemId, quantity)
description, price, total
endSale
total with taxes
return value(s)
(optional if nothing is returned)
makePayment (amount)
change due, receipt
Basic SSD

Example of an SSD for the Process Sale Scenario

Use Case: Process Sale Scenario – Main Success Story

1. Cashier starts new sale
2. Cashier enters item identifier
3. System records sale line item and presents item description, price and running total – **steps 2 and 3 are repeated until all items are processed**
4. System presents total with taxes calculated
5. Cashier tells Customer the total and asks for payment
6. Customer pays and System handles payment

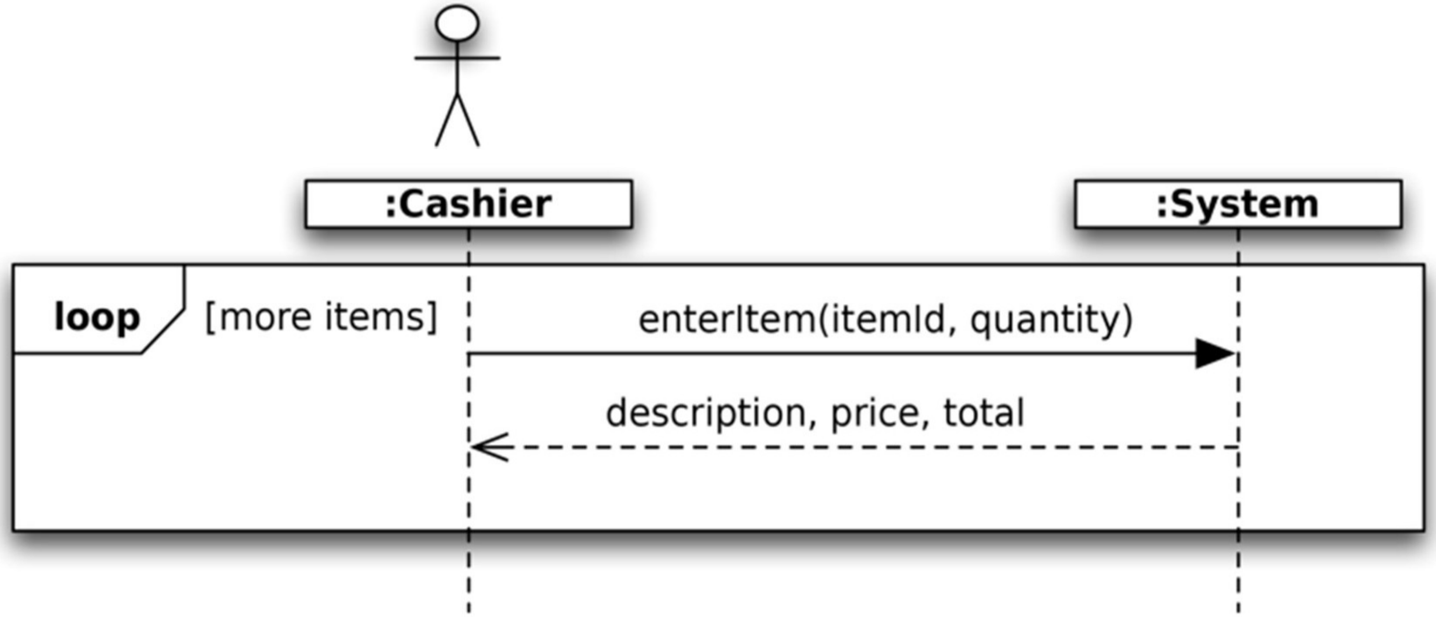
Example of an SSD for the Process Sale Scenario

Use Case: Process Sale Scenario – Main Success Story

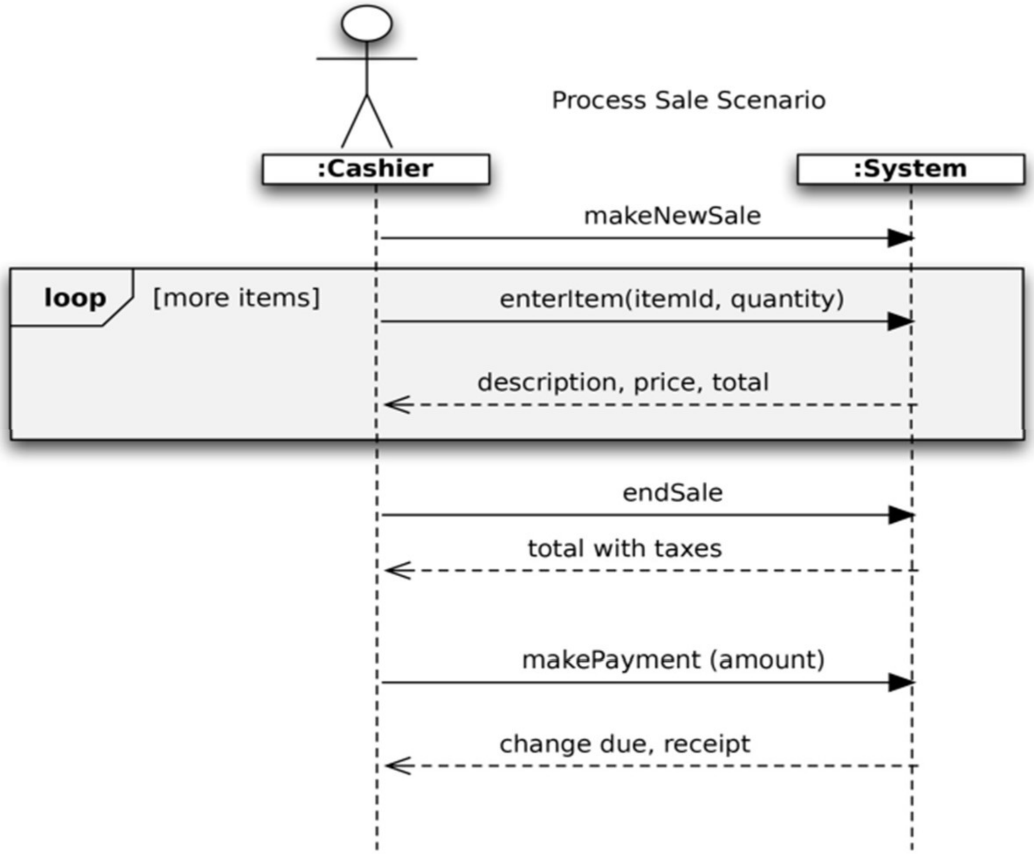
1. Cashier starts new sale
2. **Cashier enters item identifier**
3. **System records sale line item and presents item description, price and running total – steps 2 and 3 are repeated until all items are processed**
4. System presents total with taxes calculated
5. Cashier tells Customer the total and asks for payment
6. Customer pays and System handles payment

Visualizing SSDs- Excerpt from the POS Domain

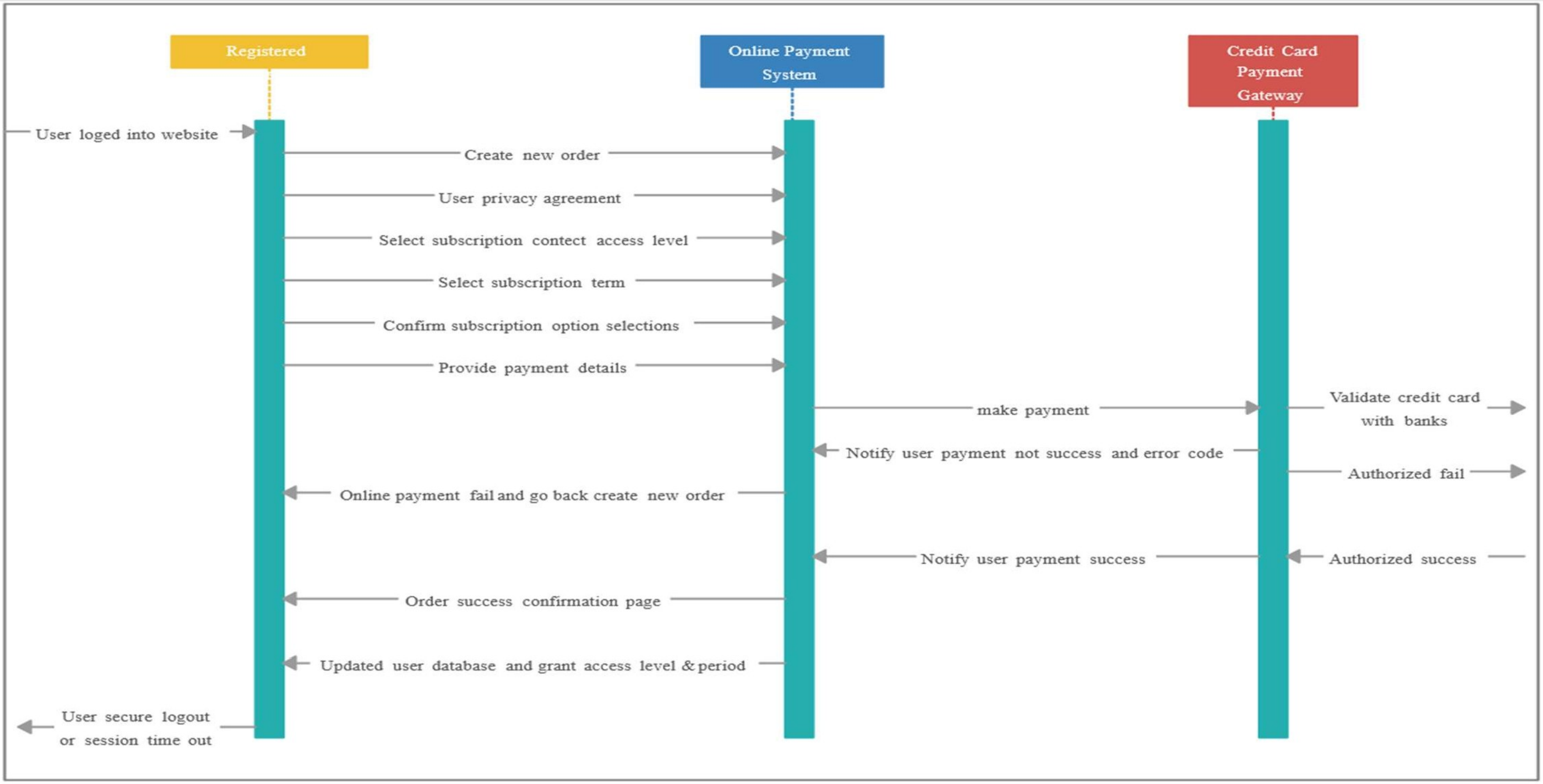
Process Sale Scenario


:Cashier
:System
loop
[more items]
enterItem(itemId, quantity)
description, price, total

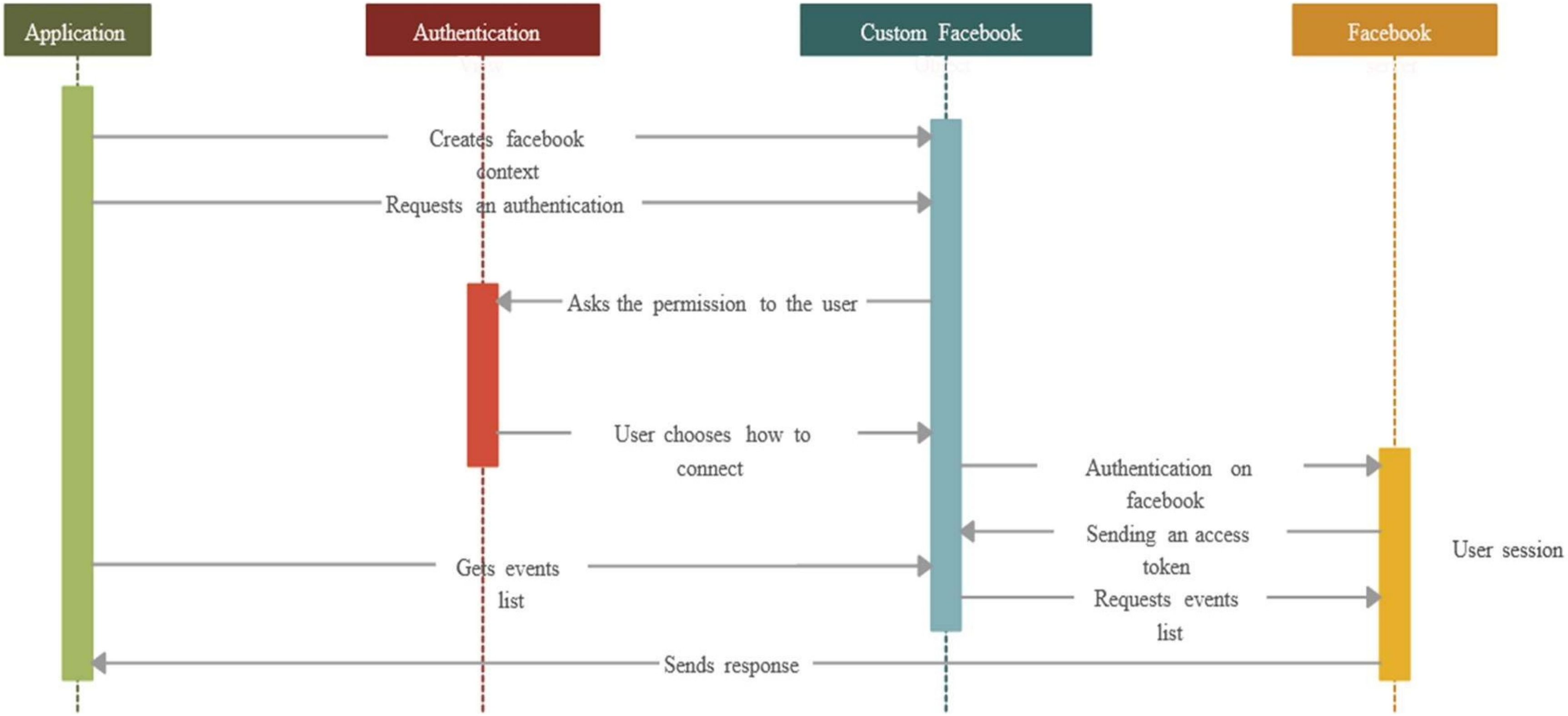
"Complete" SSD for the Process Sale Scenario


Process Sale Scenario
:Cashier
:System
makeNewSale
loop
[more items]
enterItem(itemId, quantity)
description, price, total
endSale
total with taxes
makePayment (amount)
change due, receipt

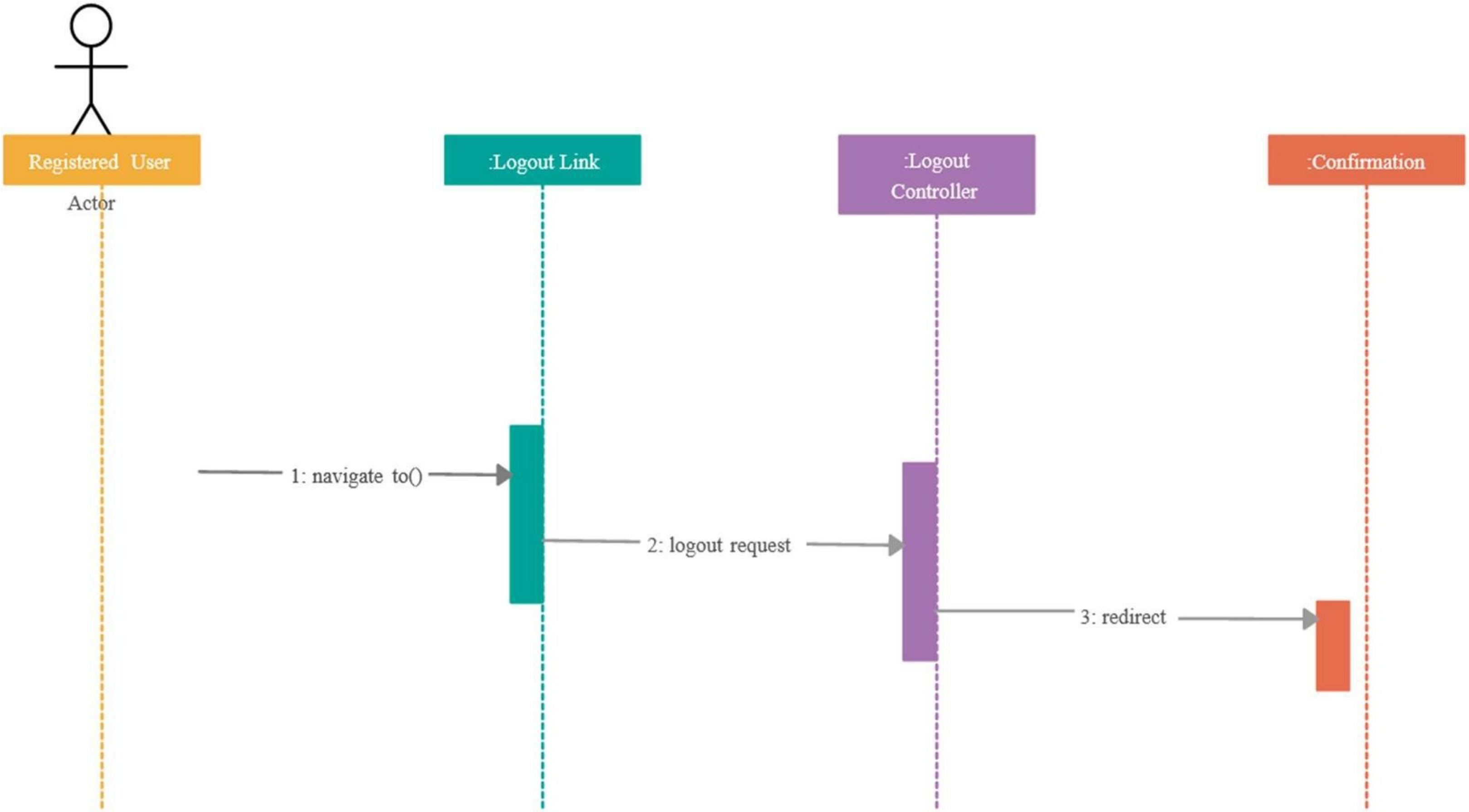
Online Payment Sequence D


Registered
Online Payment System
Credit Card Payment Gateway
User loged into website
Create new order
User privacy agreement
Select subscription contect access level
Select subscription term
Confirm subscription option selections
Provide payment details
make payment
Validate credit card with banks
Notify user payment not success and error code
Authorized fail
Online payment fail and go back create new order
Notify user payment success
Authorized success
Order success confirmation page
Updated user database and grant access level & period
User secure logout or session time out

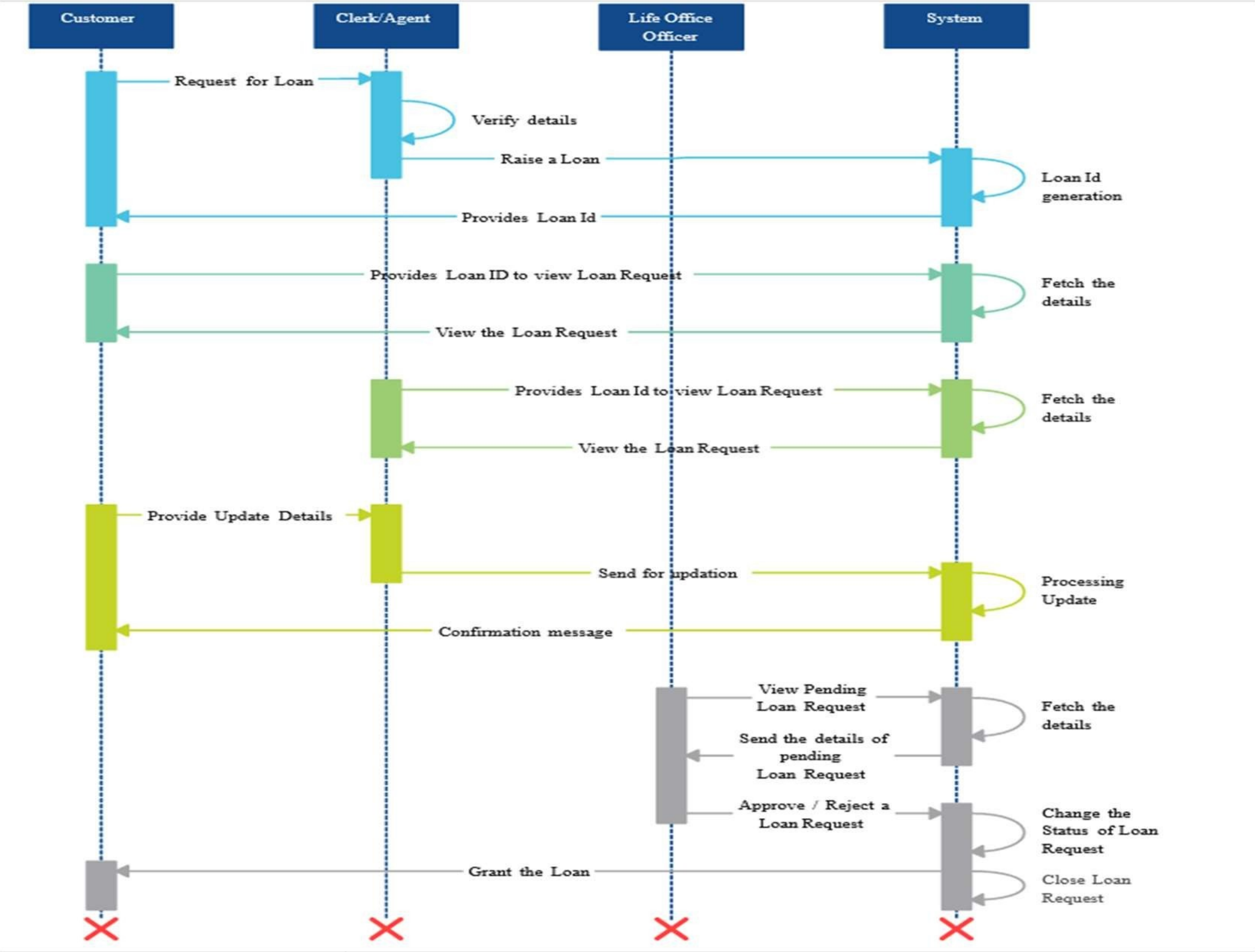
System Sequence for Facebook Authentication


Application
Authentication
Custom Facebook
Facebook
Creates facebook context
Requests an authentication
Asks the permission to the user
User chooses how to connect
Authentication on facebook
Sending an access token
Gets events list
Requests events list
User session
Sends response

Logout Sequence


Registered User
Actor
:Logout Link
:Logout Controller
:Confirmation
1: navigate to()
2: logout request
3: redirect

Loan Process System Sequence


Customer
Clerk/Agent
Life Office Officer
System
Request for Loan
Verify details
Raise a Loan
Loan Id generation
Provides Loan Id
Provides Loan ID to view Loan Request
Fetch the details
View the Loan Request
Provides Loan Id to view Loan Request
Fetch the details
View the Loan Request
Provide Update Details
Send for updation
Processing Update
Confirmation message
View Pending Loan Request
Fetch the details
Send the details of pending Loan Request
Approve / Reject a Loan Request
Change the Status of Loan Request
Grant the Loan
Close Loan Request

The Implementation of the Book Store

- It is the system sequence diagram of a book store.

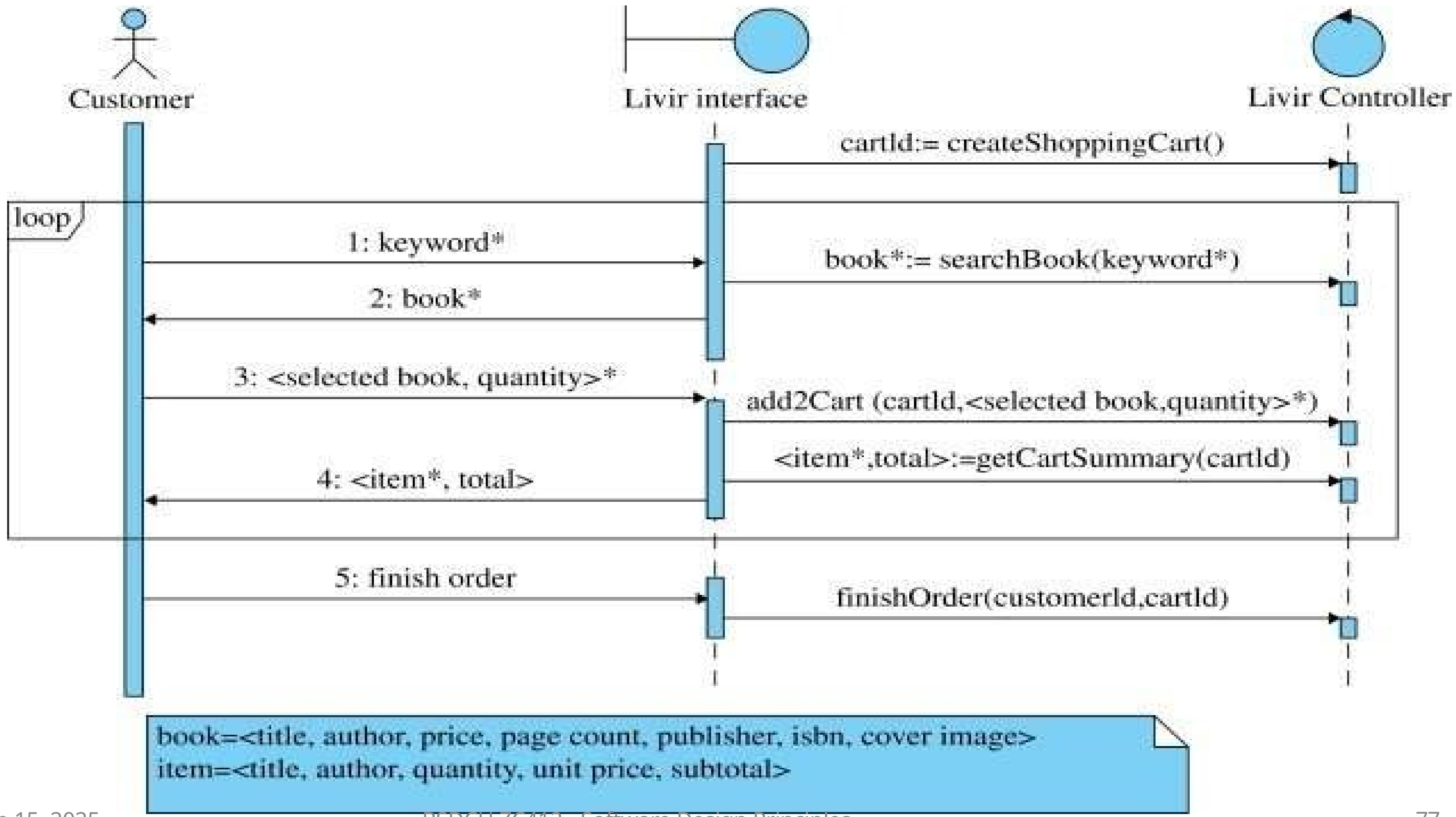
- The actor is the customer that passes his information to the interface just one time.

- The interface passes this information every time as an argument to the controller each time a system command or query needs that information.

- See, when the methods add2Cart, getCartSummary, and finish order are called, the interface passed the cartId to the controller three times.

- Now, we get the developers have made the shifts from analysis to design.

The Implementation of the Book Store


Customer
Livir interface
Livir Controller
cartId:= createShoppingCart()
loop
1: keyword*
book*:= searchBook(keyword*)
2: book*
3: <selected book, quantity>*
add2Cart (cartId,<selected book,quantity>*)
<item*,total>:=getCartSummary(cartId)
4: <item*, total>
5: finish order
finishOrder(customerId,cartId)
book=<title, author, price, page count, publisher, isbn, cover image>
item=<title, author, quantity, unit price, subtotal>

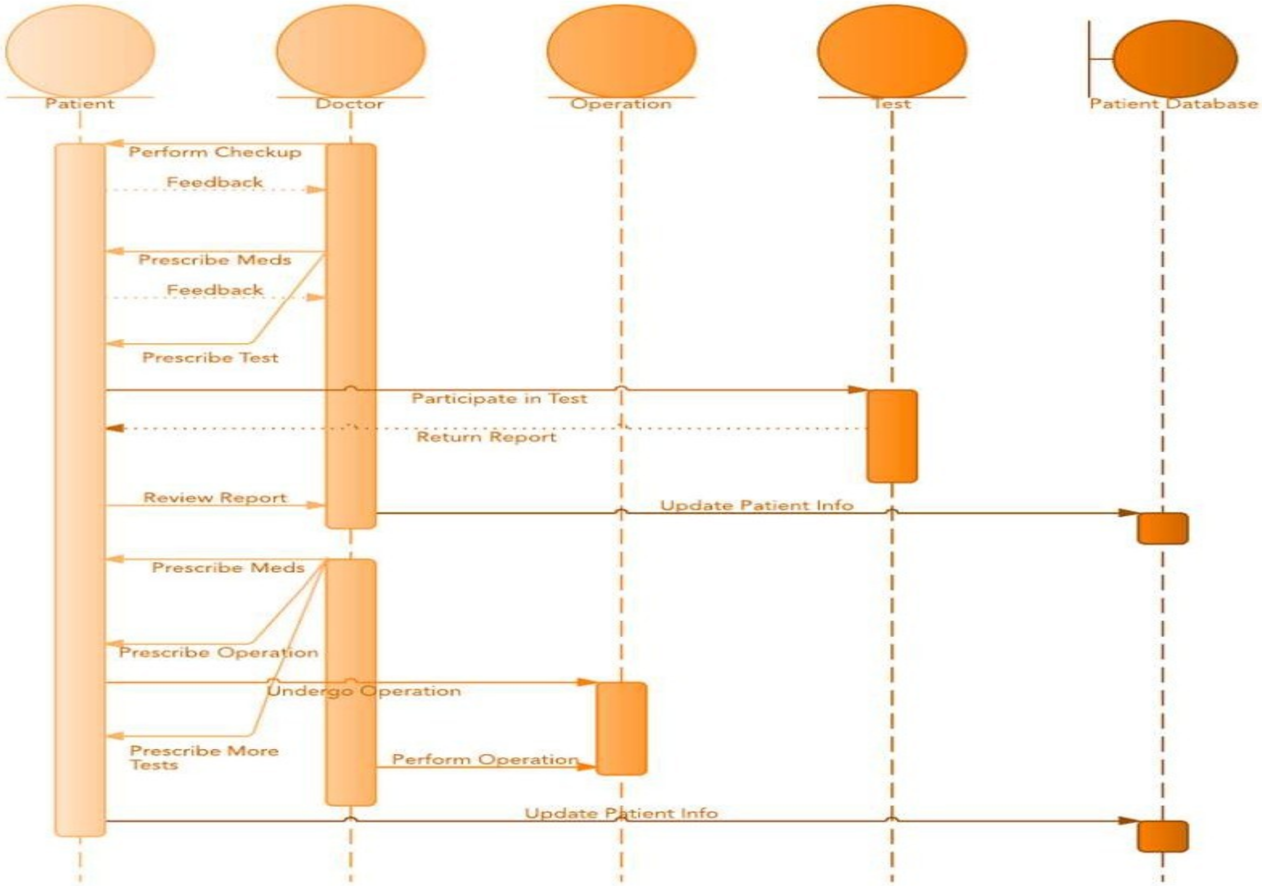
Hospital Information System

- A hospital information system, helps doctors, administrators, and hospital staff managing all of the activities and information collected at a hospital.

- including checkups, prescriptions, appointments, and information on the patients and their caretakers.

- The diagram below provides a simple view of how the primary processes operate with each other over time.

Hospital Information System


Patient
Doctor
Operation
Test
Patient Database
Perform Checkup
Feedback
Prescribe Meds
Feedback
Prescribe Test
Participate in Test
Return Report
Review Report
Update Patient Info
Prescribe Meds
Prescribe Operation
Undergo Operation
Prescribe More Tests
Perform Operation
Update Patient Info

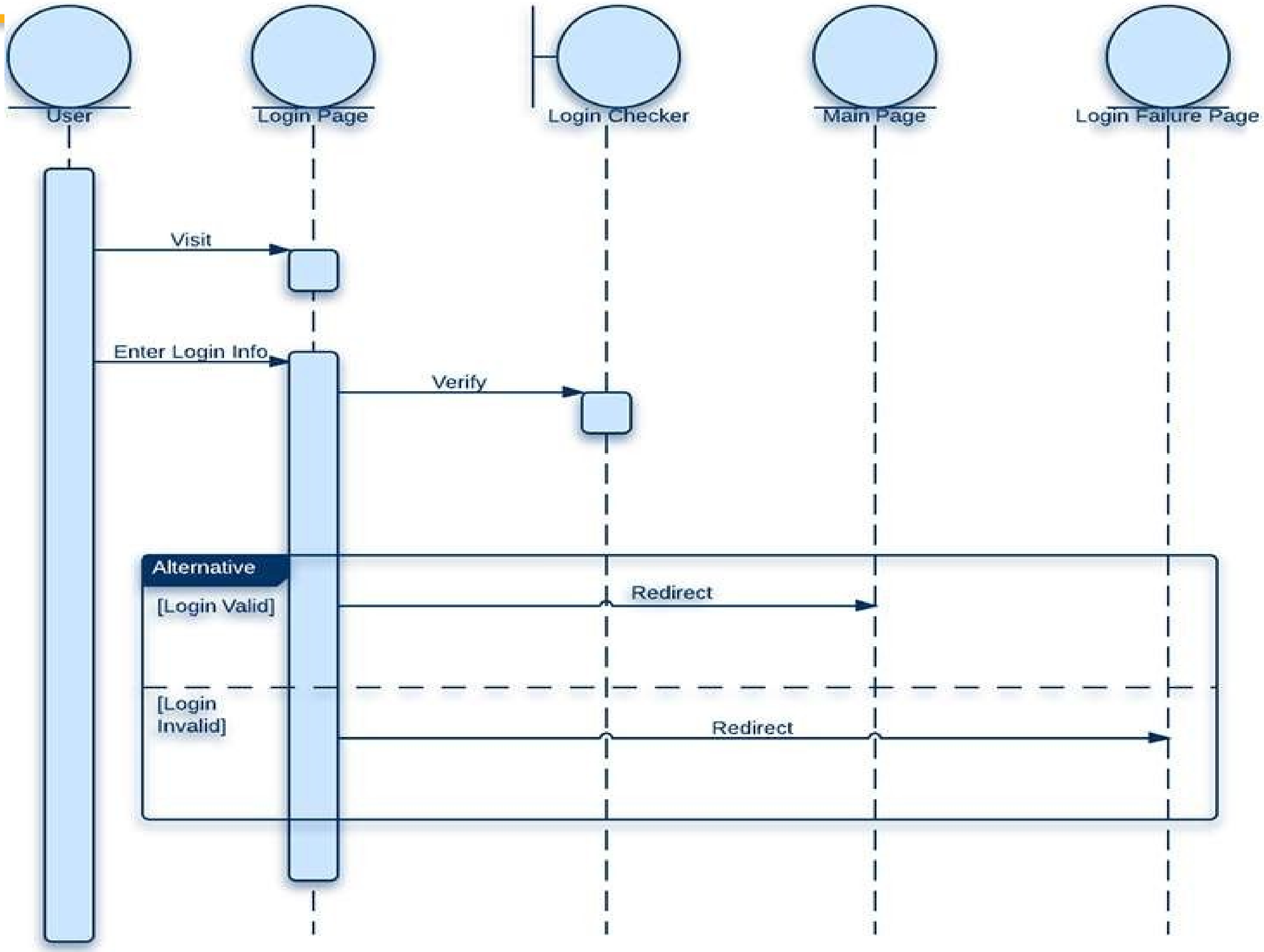
Example

- In a login sequence, two of the objects would be the potential visitor or user and the login page a visitor uses to interact with the platform they’re attempting to log into.

- The other set of components you must identify are the messages the objects will send.

- What information is passed from one object to another?

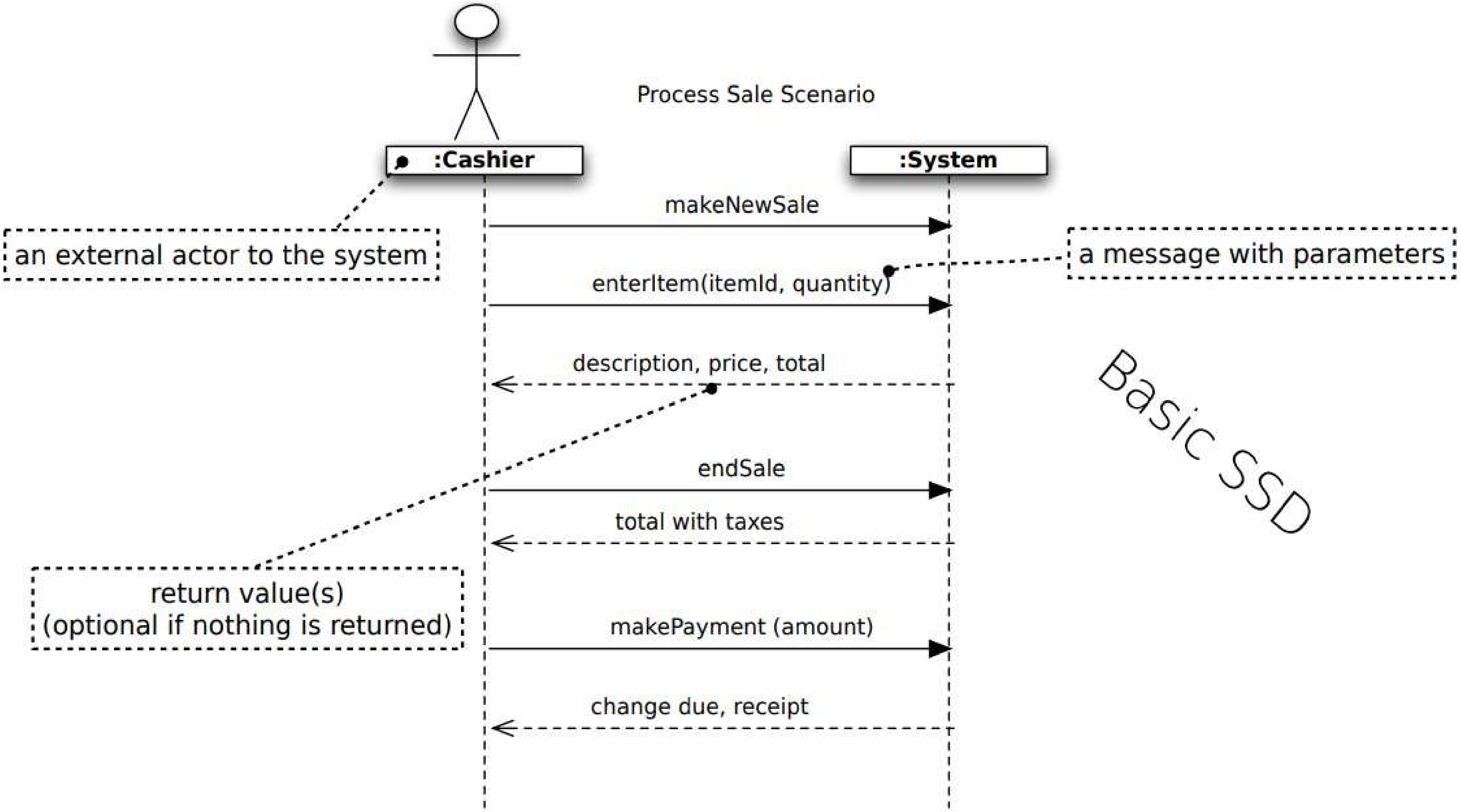
- In the user authentication process we describe, the messages show how a user enters information into the browser to gain access to the webpage and how the web browser will then cross-check credential to the designated application.


User
Login Page
Login Checker
Main Page
Login Failure Page
Visit
Enter Login Info
Verify
Alternative
[Login Valid]
Redirect
[Login Invalid]
Redirect

The Implementation of the Cashier and the System

- The cashier that is an actor starts the new sale.
- Then the actor enters the item identifier, and then the system monitors the sale line item and presents item description, price, and running total.
- The system produces the total with all the taxes calculated.
- Then the information is told to the customer, and the payment is received, and in last, the system is handling the payment.

The Implementation of the Cashier and the System


Process Sale Scenario
:Cashier
:System
an external actor to the system
makeNewSale
a message with parameters
enterItem(itemId, quantity)
description, price, total
endSale
total with taxes
return value(s)
(optional if nothing is returned)
makePayment (amount)
change due, receipt
Basic SSD

Advantages of using the System Sequence Diagram

- The system sequence diagram depicts the control flow of use cases and the events related to them.

- We can track the logic of procedures, functions, or operations with the system sequence diagram.

- It makes us understand how the control is flowing by the interaction of the actor and a system.

- It helps us in making any transition in your design.

- It is very inefficient to make the **System Sequence Diagram** by the hands because it is vulnerable to many deadly mistakes.

- When your system becomes more complex as the objects and actors increase, it isn't elementary to do this task by hand.

UML diagram summary

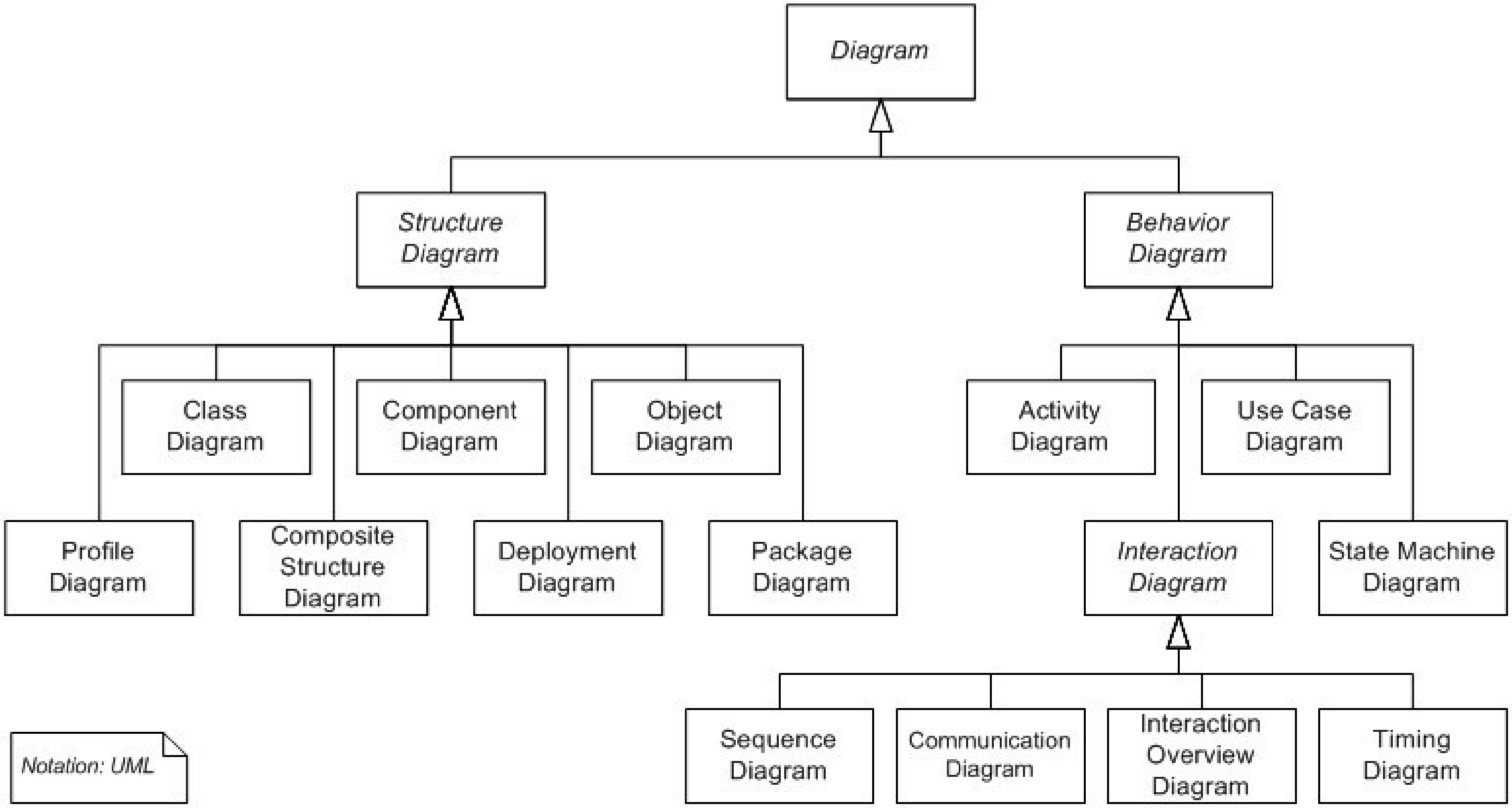
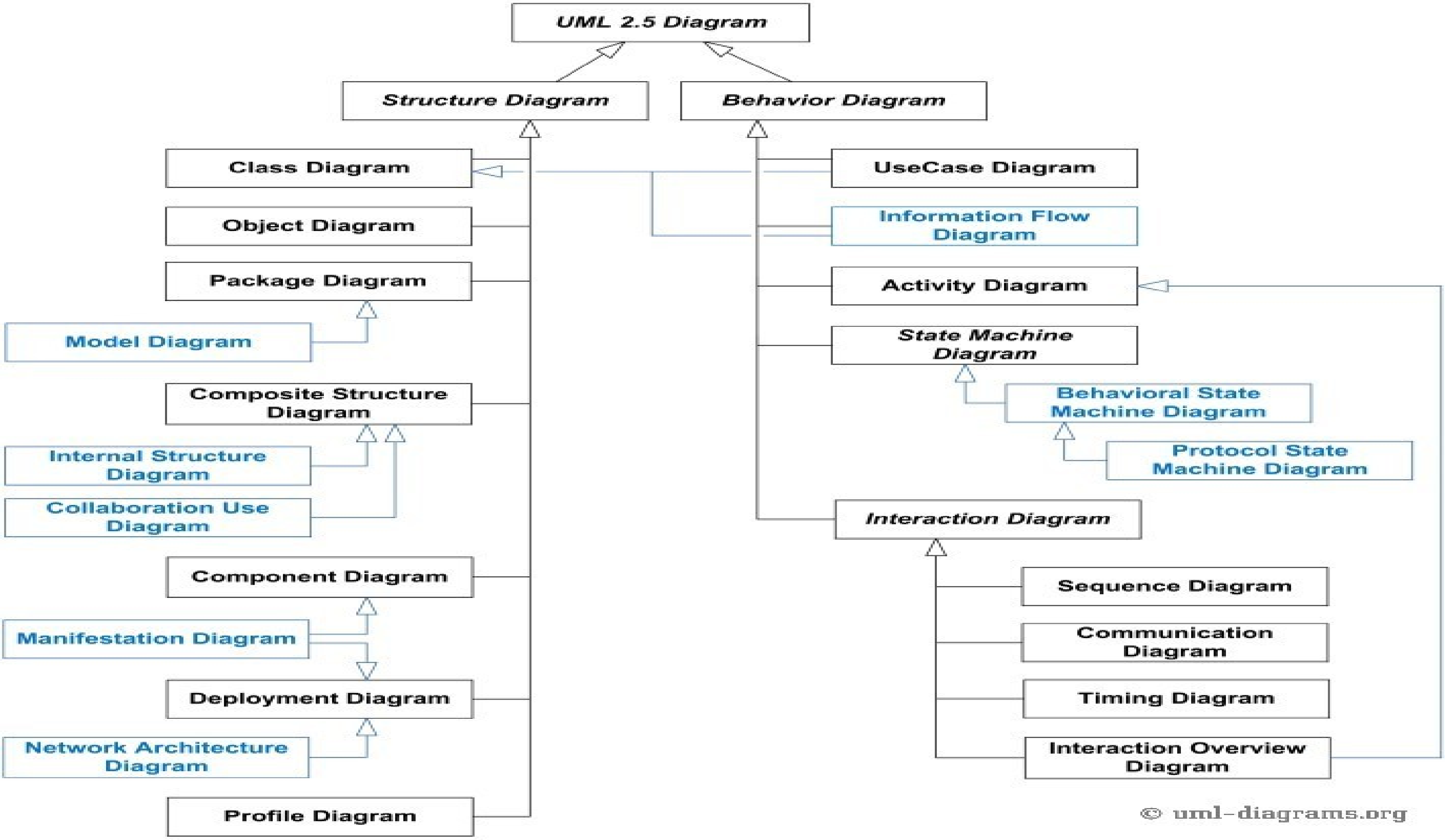


Diagram
Structure Diagram
Behavior Diagram
Class Diagram
Component Diagram
Object Diagram
Profile Diagram
Composite Structure Diagram
Deployment Diagram
Package Diagram
Activity Diagram
Use Case Diagram
Interaction Diagram
State Machine Diagram
Sequence Diagram
Communication Diagram
Interaction Overview Diagram
Timing Diagram
Notation: UML


UML 2.5 Diagram
Structure Diagram
Behavior Diagram
Class Diagram
Object Diagram
Package Diagram
Model Diagram
Composite Structure Diagram
Internal Structure Diagram
Collaboration Use Diagram
Component Diagram
Manifestation Diagram
Deployment Diagram
Network Architecture Diagram
Profile Diagram
UseCase Diagram
Information Flow Diagram
Activity Diagram
State Machine Diagram
Behavioral State Machine Diagram
Protocol State Machine Diagram
Interaction Diagram
Sequence Diagram
Communication Diagram
Timing Diagram
Interaction Overview Diagram
© uml-diagrams.org

BITS Pilani
Pilani Campus

SOFTWARE DESIGN PRINCIPLES

Unified Modeling Language for Design

By Prof A R Rahman

Identification of Software classes

Introduction

- In object-oriented software design (OOD), classes are templates for defining the characteristics and operations of an object.
- Often, classes and objects are used interchangeably, one synonymous with the other.
- In actuality, a class is a specification that an object implements.
- Identifying classes can be challenging.
- Poorly chosen classes can complicate the application’s logical structure, reduce reusability, and hinder maintenance.
- A brief overview of object-oriented classes and offers tips and suggestions to identify cohesive classes.

Classes

- Object-oriented classes support the object-oriented principles of abstraction, encapsulation, polymorphism and reusability.
- They do so by providing a template, or blueprint, that defines the variables and the methods common to all objects that are based on it.
- Classes specify knowledge (attributes) - *they know things* - and behavior (methods) - *they do things*.
- Classes are specifications for objects.
- Derived from the Use Cases, classes provide an abstraction of the requirements and provide the internal view of the application.

Design Classes

- The Requirements Model defines a set of analysis classes. Each describes some element of the problem domain, focus on an aspect of the problem that is visible.

- The level of abstraction of the analysis class is comparatively high.

- The set of design classes refine analysis classes and providing design detail that enables classes to execute a software infrastructure that supports business solutions.

Design Classes

- There are 5 different types of design classes that represent a different layer of design architecture that can be developed:

1. User interface classes define abstraction that mandatory for human-computer interaction [HCI]. In cases, HCI occurs within the context of metaphor, and design classes for the interface may be visible representations of elements of metaphor.
2. Business domain classes are often refinements of analysis classes defined earlier. The class identifies the attributes that are required to implement some elements of the business domain.
3. Process classes implement lower-level business Preoccupation need to manage business domain classes.

Design Classes

4. **Persistent classes** represent the data stores that will persist beyond the execution of software.

5. **System classes implement** software management and control function that permits the system to operate and convey within its computing environment and with the outside world

- The level of abstraction is reduced as each analysis class transformed into to design representations.
- That is, analysis classes represent data objects using the jargon of the business domain. Design classes present notably more technical detail as a guide for implementation.
- **Arlow** and **Neustadt** suggest each design class reviewed to ensure that is "well-formed".
- They define four characteristics of a well-formed design class:

Characteristics:

1. Complete and sufficient: A design class should be complete encapsulation of all attributes and method that can be reasonably be expected to exist for class.

2. For example, the class scene defined for video editing software is complete only if it contains all attributes and methods that can agreeably be associated with the creation of a video scene.

3. Sufficiently ensure that design class contains only those methods that are sufficient to achieve the intent of class, no more and no less.

Primitiveness:

- Method associate with design class should be focused on accomplishing one service for class.

- Once service implemented with the method, the class should not provide another way to accomplish the same thing.

- For example, the class Video Clip for video editing software might have attributes *start point* and *end point* to specify start and endpoint of clip.

High Cohesion

- High Cohesion :<strong>

- A cohesion design class has a small, concentrated set of authority and single-mindedly applies attributes and methods to implement those responsibilities.

- For example, the class video clip might contain set of method for editing the video clip.

- As long as each method focus solely on attributes associated with video clip, cohesion is maintained.

Low Coupling

- Low Coupling : Within the design model, it is necessary for design classes to get together with one another.
- However, get together should be kept to an acceptable minimum.
- If the design model is highly coupled, the system is difficult to implement to test and to maintain over time.
- In general, design classes within subsystem should have only limited knowledge of other classes.
- This restriction called the Law of Demeter, suggest that method should only send message to methods in neighboring classes.

Attributes

- Attributes define the characteristics of the class that, collectively, capture all the information about the class.

- Attributes should be protected by their enclosing class.

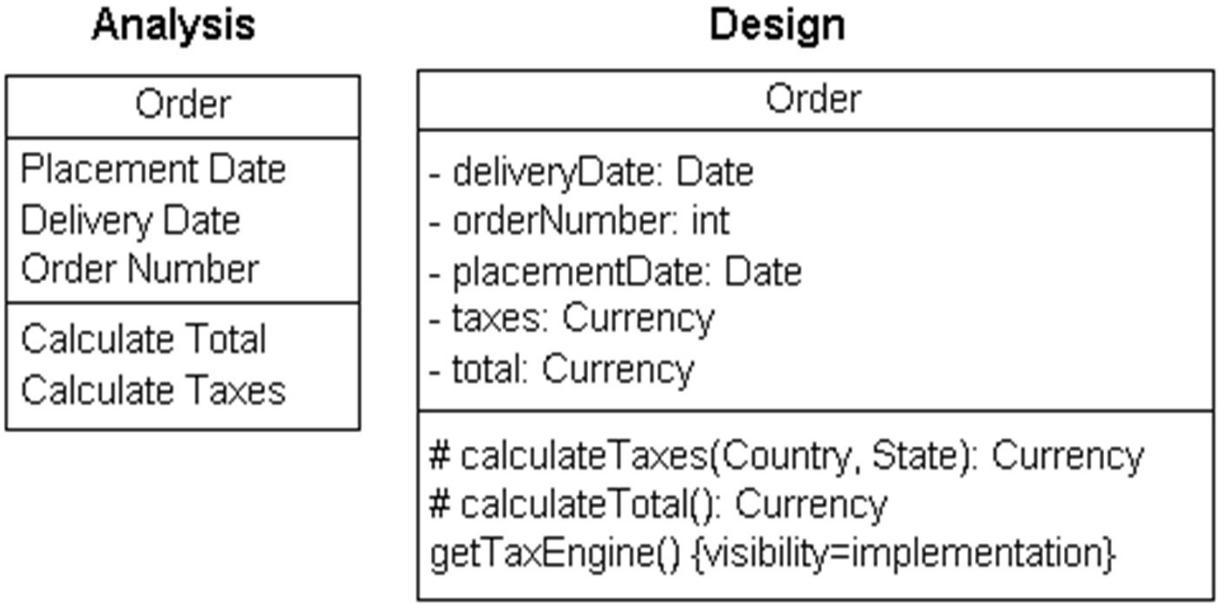
- Unless changed by the class behavior, attributes maintain their values.

- The type of data that an attribute can contain is determined by its data type.

General Guidelines

- Class diagrams are used for a variety of purposes - from understanding requirements to describing your detailed design - you will need to apply a different style in each circumstance.

- This section describes style guidelines pertaining to different types of class diagrams.


Analysis
Order
Placement Date
Delivery Date
Order Number
Calculate Total
Calculate Taxes
Design
Order
- deliveryDate: Date
- orderNumber: int
- placementDate: Date
- taxes: Currency
- total: Currency
# calculateTaxes(Country, State): Currency
# calculateTotal(): Currency
getTaxEngine() {visibility=implementation}

Analysis and design versions of a class.

Class Style Guidelines

- A class is effectively a template from which objects are created (instantiated).

- Although in the real world Doug, Wayne, John, and Bill are all student objects we would model the class *Student* instead.

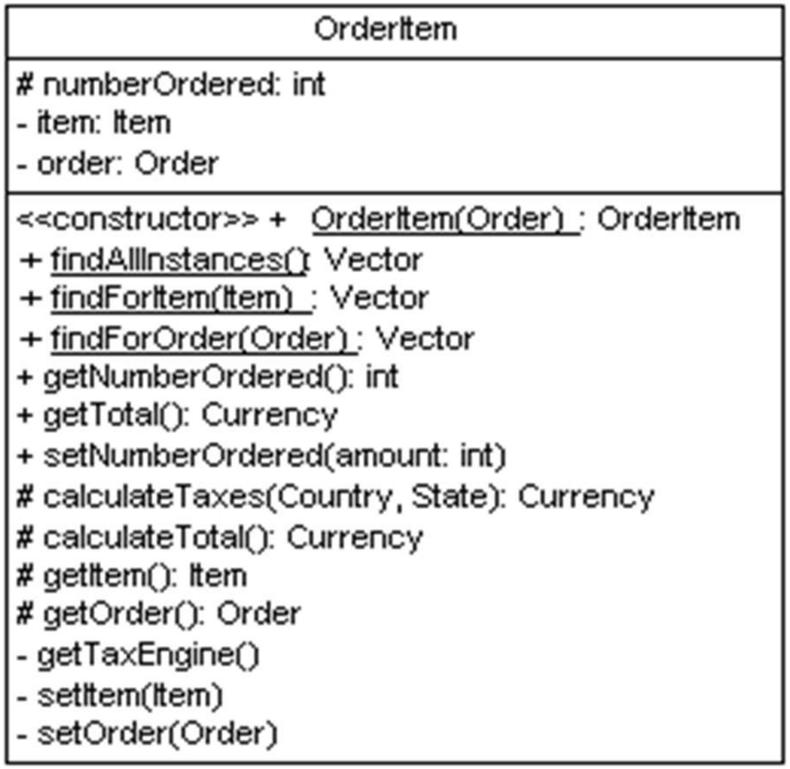
- Classes define attributes, information that is pertinent to their instances, and operations, functionality that the objects support.

- Classes will also realize interfaces (more on this later).

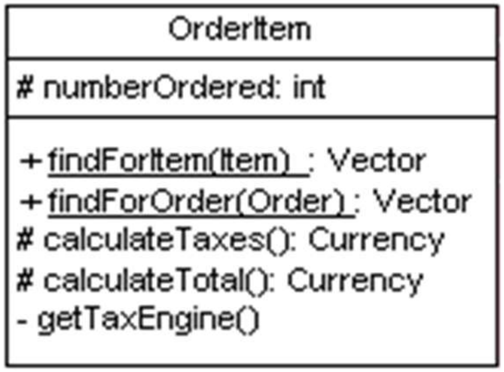
- Note that you may need to soften some of the naming guidelines to reflect your implementation language or software purchased from a third-party vendor.

The OrderItem class with and without scaffolding code

With Scaffolding


OrderItem
# numberOrdered: int
- item: Item
- order: Order
<<constructor>> + OrderItem(Order) : OrderItem
+ findAllInstances(): Vector
+ findForItem(Item) : Vector
+ findForOrder(Order) : Vector
+ getNumberOrdered(): int
+ getTotal(): Currency
+ setNumberOrdered(amount: int)
# calculateTaxes(Country, State): Currency
# calculateTotal(): Currency
# getItem(): Item
# getOrder(): Order
- getTaxEngine()
- setItem(Item)
- setOrder(Order)

Without Scaffolding


OrderItem
# numberOrdered: int
+ findForItem(Item) : Vector
+ findForOrder(Order) : Vector
# calculateTaxes(): Currency
# calculateTotal(): Currency
- getTaxEngine()

Indicating the exceptions thrown by an operation

- findAllInstances(): Vector {exceptions=NetworkFailure, DatabaseError}

1. Use Common Terminology for Names

2. Prefer Complete Singular Nouns for Class Names

3. Name Operations with a Strong Verb

Indicating the exceptions thrown by an operation

1. Name Attributes With a Domain-Based Noun

2. Do Not Model Scaffolding Code.

3. Scaffolding code refers to the attributes and operations required to implement basic functionality within your classes, such as the code required to implement relationships with other classes.

4. Figure depicts the difference between the *OrderItem* class without scaffolding code and with it.

Best Practices

1. Never Show Classes With Just Two Compartments

2. Label Uncommon Class Compartments

3. Include an Ellipsis (. . .) At The End of Incomplete Lists

4. List Static Operations/Attributes Before Instance Operations/Attributes

5. List Operations/Attributes in Decreasing Visibility

Best Practices

1. For Parameters That Are Objects, Only List Their Type

2. Develop Consistent Method Signatures

3. Avoid Stereotypes Implied By Language Naming Conventions

4. Indicate Exceptions In An Operation's Property String.

5. Exceptions can be indicated with a UML property string, an example of which is shown in Figure.

There are two basic data types: Primitive and Derived.

1.Primitive data types are fundamental types. Examples are integer, string, float.

2.Derived data types are defined in terms of the Primitive data types, that is, they form new data types by extending the primitive data types.

3.A Student class, for example, is a derived data type formed by a collection of primitive data types.

When defined in the context of a problem domain, derived data types are called Domain Specific Data types.

These are the types that define and constrain attributes to be consistent with the semantics of the data.

For example, Address studentAddress versus string studentAddress.

A practical - design models

A practical, design models should use domain specific data types in lieu of primitive data types

When looking for attributes in the design model, look for these types:

1. Descriptive attributes provide intrinsic information about the class.

For example, information about enrollment status, test scores, and education goals would be intrinsic to a Student class but not an Automobile class.

Ask, “What characteristics distinguish this class from others?”

A practical - design models

2.Naming attributes are used to uniquely identify an object and typically don’t change during the lifetime of the object.

Consider two Student objects that contain the naming attribute, studentName.

The values of “Jack” and “Jill” would identify two different Student objects.

Ask, “What uniquely identifies this object from another object of the same class?”

A practical - design models

3. Referential attributes contain links to objects.

A student’s test scores might be stored in an Assessment class, for example.

The value for the Assessment attribute (in the Student class) would link the Student class to the Assessment class.

Ask, “How is this object associated with other objects?”

Responsibilities

- A class has behaviors and responsibilities.
- A responsibility is something that a class knows or does…it's an obligation a class has to know certain information and/or to perform a task.
- For example, a student knows about her address and Aadhar/SSN.
- A student does enroll in a class.
- Technically, responsibilities are not the same as the class' operations (behavior), but, operations fulfill the responsibilities.
- It may take more than one operation (sometimes, more than one object) to fulfill the responsibility.
- Responsibility is the obligation of a class or object to perform a task or know information.

Identifying classes

- Identifying object-oriented classes is both a skill and an art.
- It’s a process that one gets better at over time.
- For example, it’s not unusual for inexperienced designers to identify too many classes.
- Modeling too many classes results in poor performance, unnecessary complexity and increased maintenance.
- On the other hand, too few classes tend to increase couplings, and make classes larger and unwieldy.
- In general, strive for class cohesiveness where behavior is shared between multiple, related classes rather than one very large class.

Cohesive classes

- Cohesive classes reduce coupling, enable extensibility and increase maintainability.

- Classes that seem obvious wind up being poor choices and classes that are initially hidden or that rely on problem domain knowledge wind up as the best choices.

- Begin class modeling by identifying candidate classes - an initial list of classes from which the actual design classes will emerge.

Cohesive classes

- Design classes are the classes that are modeled.

- Candidate classes exist for the sole purpose of deriving the design classes.

- Initially, there will be a lot of candidate classes – that’s good.

- However, through analysis, their number will be reduced as they are dropped, combined and merged.

Candidate classes

- Candidate classes provide the initial impetus to produce cohesive classes.

- Candidate classes can be discovered in a variety of ways. Here are three:

1. Noun and noun phrases: Identify the noun and noun phrases, verbs (actions) and adjectives (attributes) from the Use Cases, Actor-Goal List, Application Narrative and Problem Description.

2. CRC cards: an informal, group approach to object modeling.

3. GRASP: A formal set of principles that assign responsibilities.

Candidate classes

Each of these methods will yield a list candidate classes.

The list won't be complete nor will every class be appropriate and there will likely be a mix of business and system oriented classes;

i.e.; Student and StudentRecord,

for example. That's fine. The goal is to identify the major classes - the obvious ones.

Other classes will become apparent as the design process continues.

Once the list has been created, analyze the candidate classes for associations with other classes.

Look for collaborating classes

How does each relate to each other and to the business process?

Sometimes, it’s helpful to ask, “Why keep this class?”

In other words, assume the class is redundant or unnecessary.

Keep it only if it plays a collaborating role.

Often you’ll find the class’ functionality is accomplished by another class or within the context of another class.

When a class is kept, move it to the list of design classes.

Eventually, a list of design classes will result that provides the foundational structure for the application.

Course Registration System

Candidate Classes	Design Classes
Student	Student
Teacher	Professor
Class	Course
Subject	Assessment
Grades	
Tests	

Design classes will emerge from the analysis of the candidate classes.

Associations are the key to identifying cohesive classes.

The following subsections identify the various associations that can exist between classes and suggestions to identify them.

Associations

- The classes in an application system don’t exist in a vacuum.

- Classes are associated with, or related to, other classes.

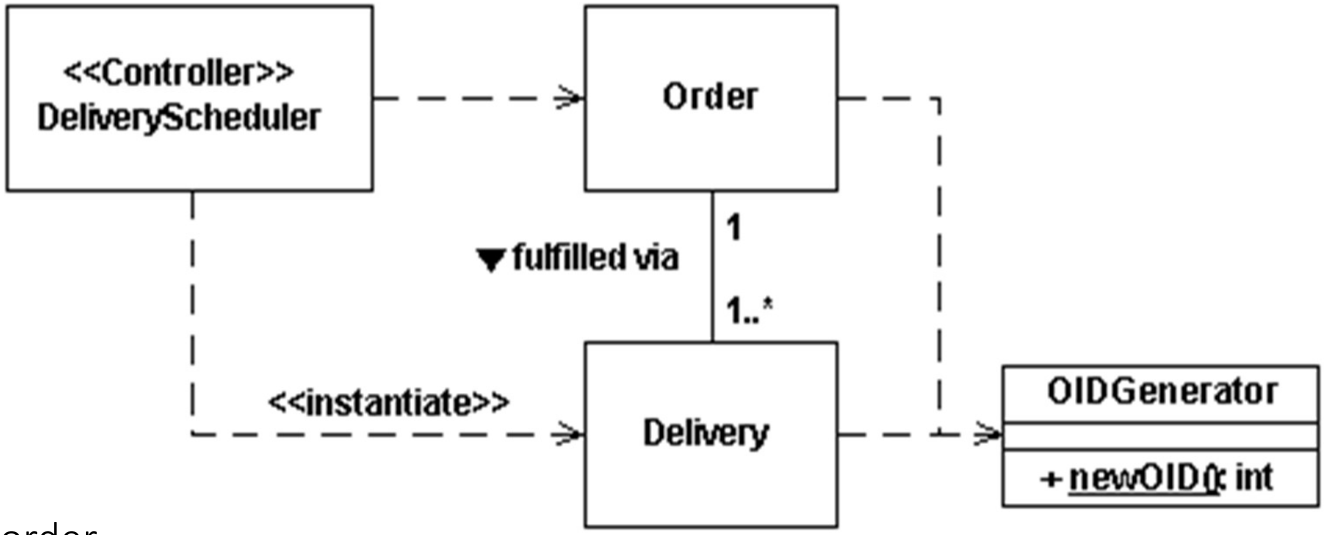
- These relationships occur when a class has, uses, knows about, or is acquainted with, one or more classes.

A relationship is an association between classes.

- When identifying relationships, start with the class that interacts with as many other classes as possible; perhaps, the core classes of the application.
- It's helpful to ask,
- Who cares about this class?
- Who is interested in this class?
- Why is this class necessary?
- Starting with the core classes will quickly identify the other relationships.

Relationship Guidelines

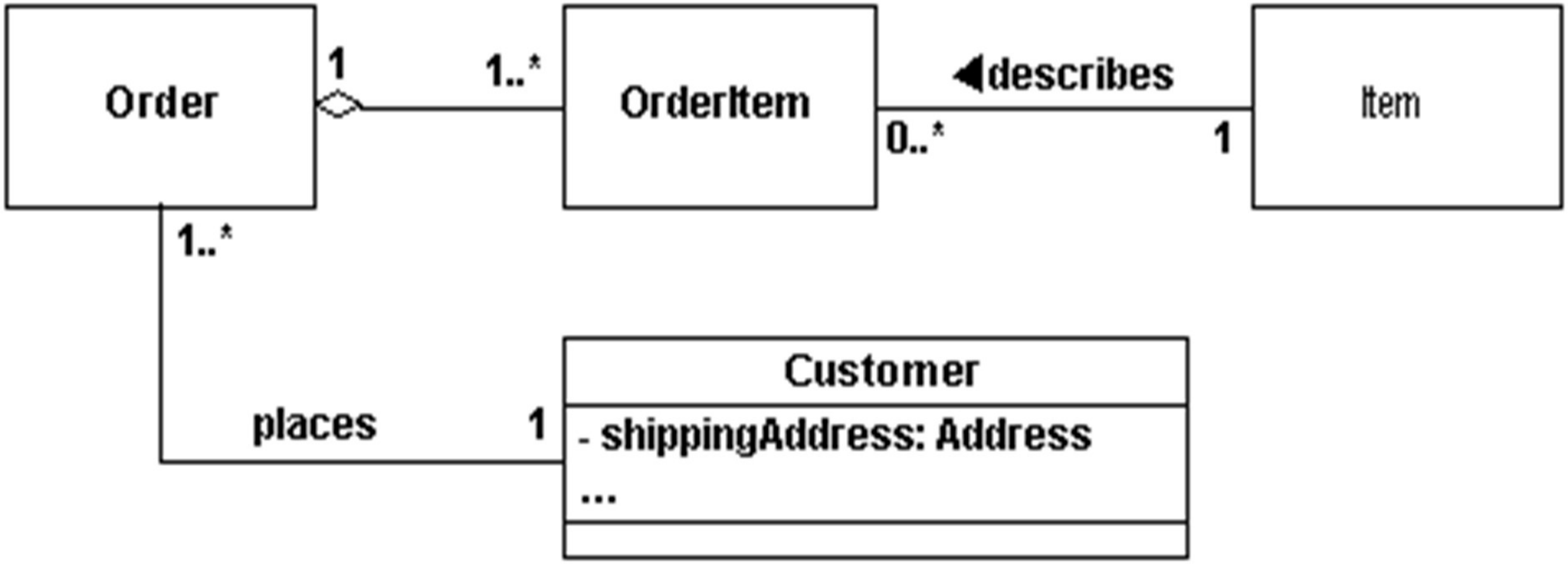
- For ease of discussion the term relationships shall include all UML concepts such as associations, aggregation, composition, dependencies, inheritance, and realizations - in other words, if it's a line on a UML class diagram we'll consider it a relationship.



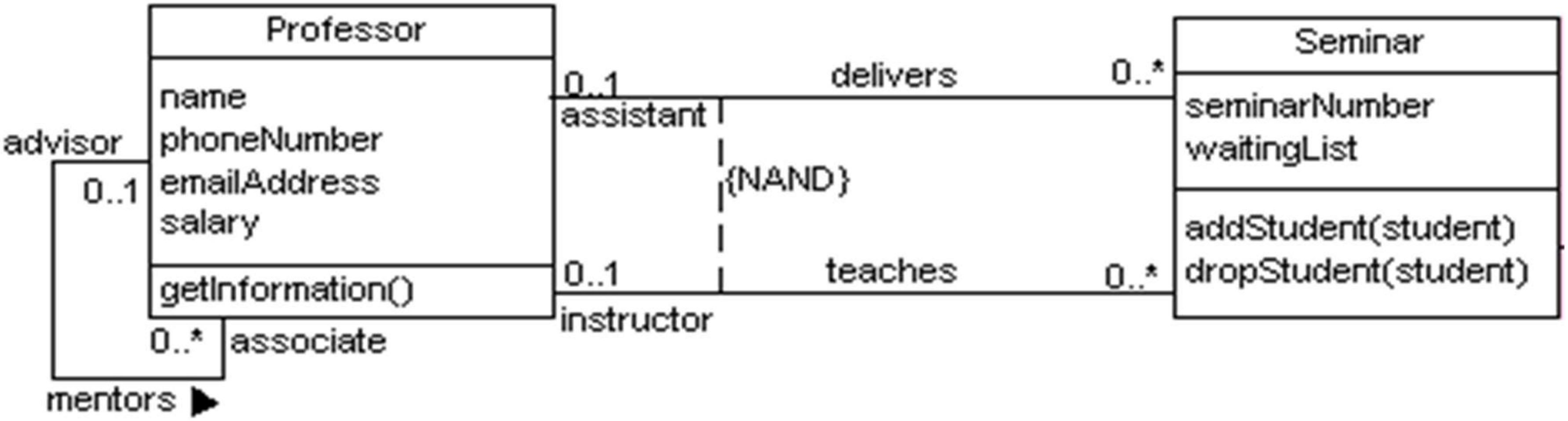
<<Controller>>
DeliveryScheduler
Order
1
▼fulfilled via
1..*
<<instantiate>>
Delivery
OIDGenerator
+ newOID(): int

Shipping an order.

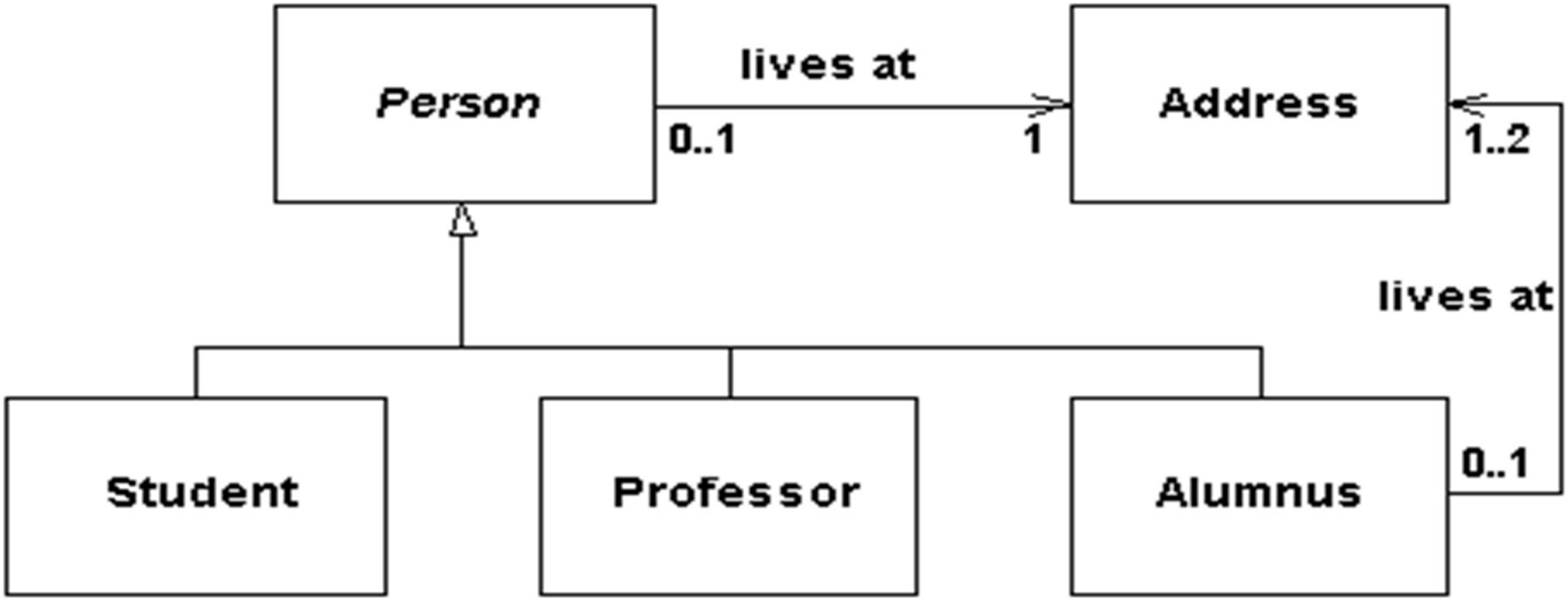
Modeling an order.


Order
1
1..*
OrderItem
◀describes
0..*
1
Item
1..*
places
1
Customer
- shippingAddress: Address
...

Professors and seminars


Professor
name
phoneNumber
emailAddress
salary
getInformation()
advisor
0..1
0..*
associate
mentors ▶
0..1
assistant
delivers
0..*
{NAND}
0..1
teaches
0..*
instructor
Seminar
seminarNumber
waitingList
addStudent(student)
dropStudent(student)

Modeling people at a university.


Person
lives at
0..1
1
Address
1..2
lives at
Student
Professor
Alumnus
0..1

1.Model Relationships Horizontally

2.Collaboration Indicates Need for a Relationship

3.Model a Dependency When The Relationship is Transitory

4.Depict Similar Relationships Involving A Common Class As A Tree.

In Figure you see that both *Delivery* and *Order* have a dependency on *OIDGenerator*.

Note how the two dependencies are drawn in combination in "tree configuration", instead of as two separate lines, to reduce clutter in the diagram.

6.Always Indicate the Multiplicity

7.Avoid a Multiplicity of "*"

7.Replace Relationships By Indicating Attribute Types.

In Figure we see that Customer has a *shippingAddress* attribute of type *Address* - part of the scaffolding code to maintain the association between customer objects and address objects.

8.Do Not Model Implied Relationships

9.Do Not Model Every Single Dependency

10.Center Names on Associations

11. Write Concise Association Names In Active Voice

12. Indicate Directionality To Clarify An Association Name

13. Name Unidirectional Associations In The Same Direction

14. Word Association Names Left-To-Right

15. Indicate Role Names When Multiple Associations Between Two Classes Exist

16.Indicate Role Names on Recursive Associations

17.Make Associations Bi-Directional Only When Collaboration Occurs In Both Directions.

18.The *lives at* association of Figure is uni-directional.

19. Redraw Inherited Associations Only When Something Changes

20. Question Multiplicities Involving Minimums And Maximums

Start with core associations.

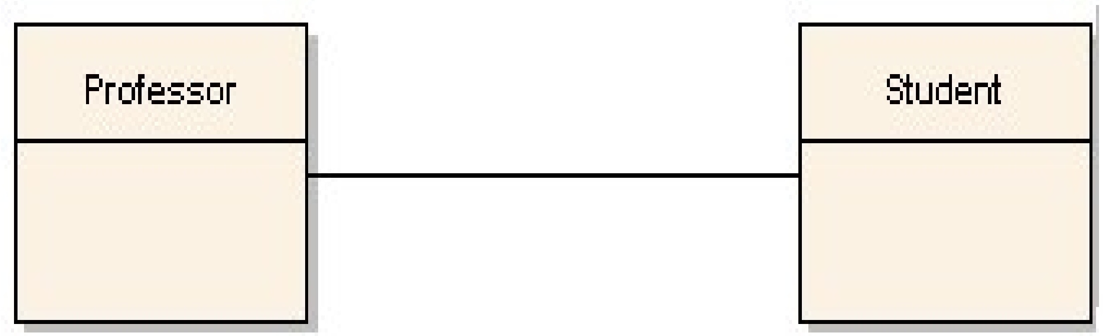


Figure 1: The Professor class is associated with the Student class

An association is usually modeled using a solid line that connects two classes.

Start with core associations.

- Most times, a single line doesn't provide enough information.

- Form a practice of giving each association a name to clarify the relationship.

- Use verbs or simple verb phrases as association names.

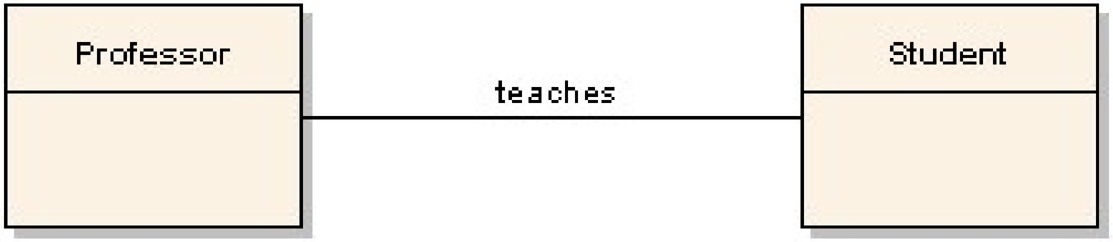


Figure 2: Professor "teaches " a Student

Association roles

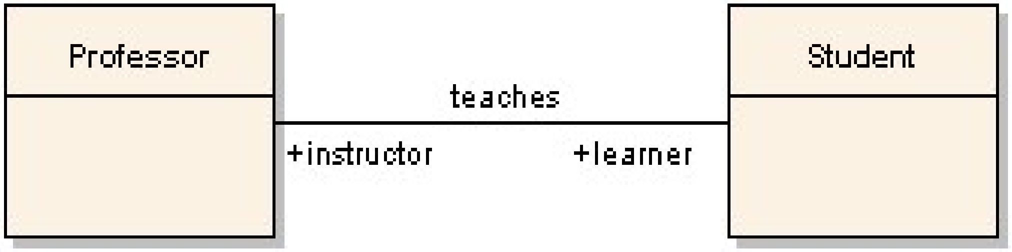

Professor
Student
teaches
+instructor
+learner

Figure 3: Association Roles

Associations also have roles.

Each class in an association has a role that describes its meaning in the relationship.

Roles are optional and if, used, should describe the role as a noun.

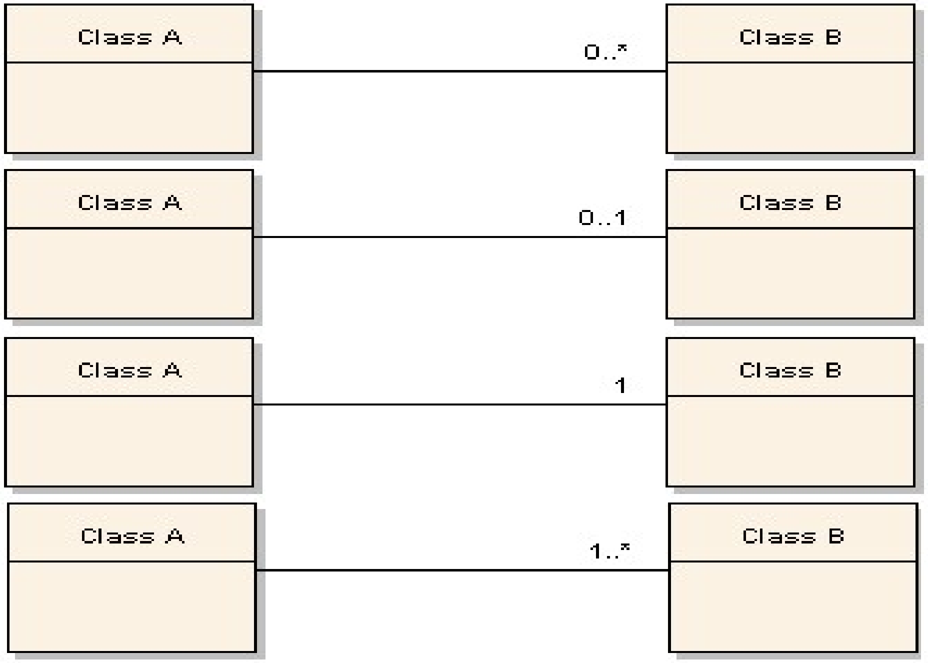
For example, a Professor is an instructor to a Student who is a learner or pupil.

Multiplicity Indicators

- A role can have multiplicity - an indication of how many objects participate in the relationship.

- Multiplicity indicators can be conditional or unconditional.

- Examples of multiplicity indicators are:


Class A
0..*
Class B
Class A
0..1
Class B
Class A
1
Class B
Class A
1..*
Class B

Objects of Class A may know about many objects of Class B

Objects of Class A may know about a single object of Class B

Objects of Class A must know about a single object of Class B

Objects of Class A must know at least one object of Class B

Figure 4: Multiplicity indicators

- In Figure 4, the first two indicators (that start with "0...") are conditional meaning no objects need be present in the relationship.
- The last two indicators (that start with "1..") are unconditional meaning at least one object must be present in the relationship. For example:

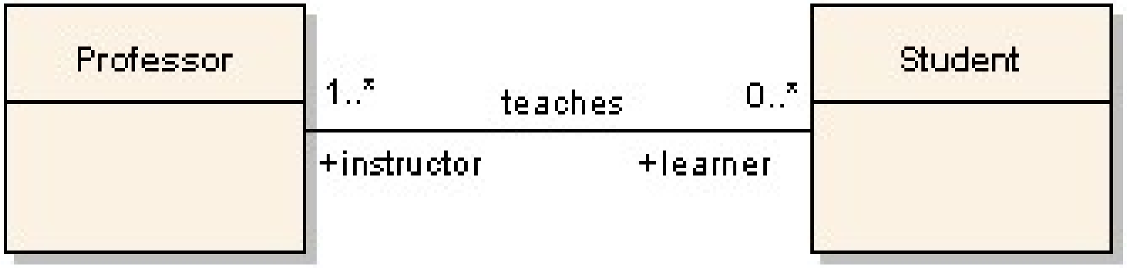


Figure 5: Conditional and Unconditional Indicators

In Figure 5, a Professor can exist without the existence of a Student.

Therefore, the multiplicity for the Student is 0…*.

However, the same is not true for the Student.

A Student must have at least one Professor. Hence, the 1…* indicator for the Professor.

Multiplicity indicators are also referred to as cardinalities.

- Unconditional indicators may impose a referential integrity constraint. Consider the following example:

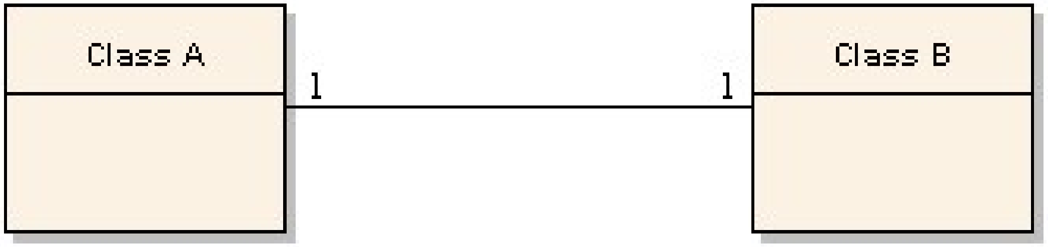


Figure 6: Special Case Condition

Class A and class B depend on the existence of the other.

Since the multiplicity is unconditional the following is implied:

1.When class A (or class B) is removed, the corresponding class B (or class A) must also be removed.

2.When class A (or class B) is added, the corresponding class B (or class A) must also be added.

Unconditional indicators must be checked for referential integrity constraints..

Association class

An association can also possess its own attributes and behavior – just like a class.

Sometimes, data exist that does not strictly belong to any of the participating classes.

In these cases, an Association class is created to map the data to the participating classes.

The class, then, becomes the association.

The association class will contain attributes that include pointers, or references, to instances of the two classes.

Association class

For example, a Student class has an association to a Course class.

A student can take many courses, and a course can be taken by many students.

However, who is responsible for the grade?

Placing the grade in the Student class gives a student the same grade for all courses.

Placing the grade in the Course class gives all students taking the same course the same grade.

The Association class resolves this issue by linking (or mapping) the grade (and other attributes) to the Student and the Course classes.

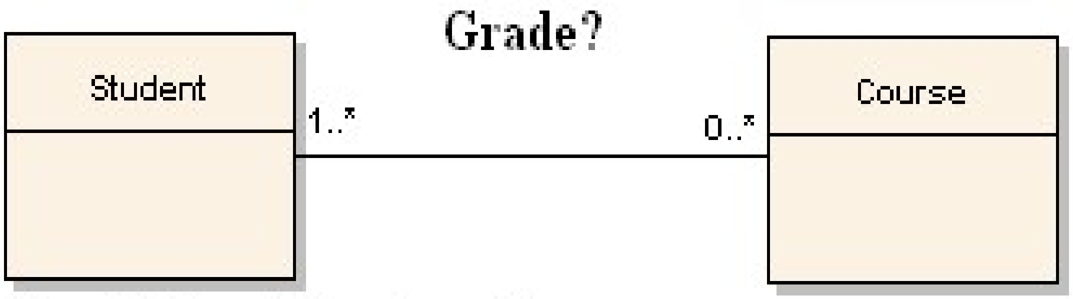


Figure 7: Where to place the grade?

A Student can have many grades for many courses,
but each grade in the StudentClassAssociation is associated to a single student and course.

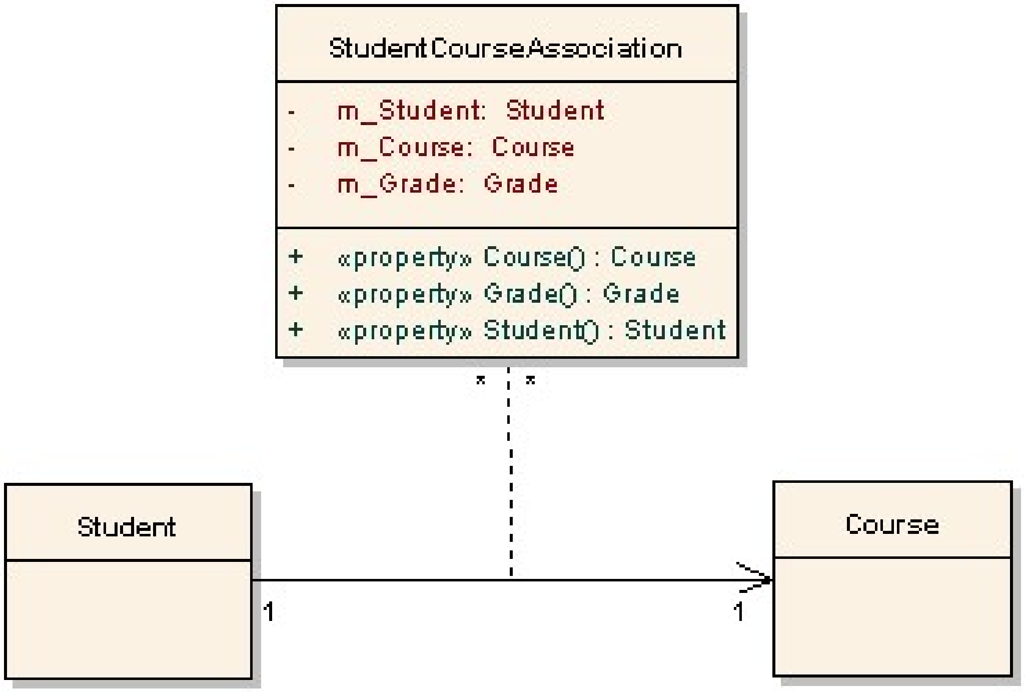
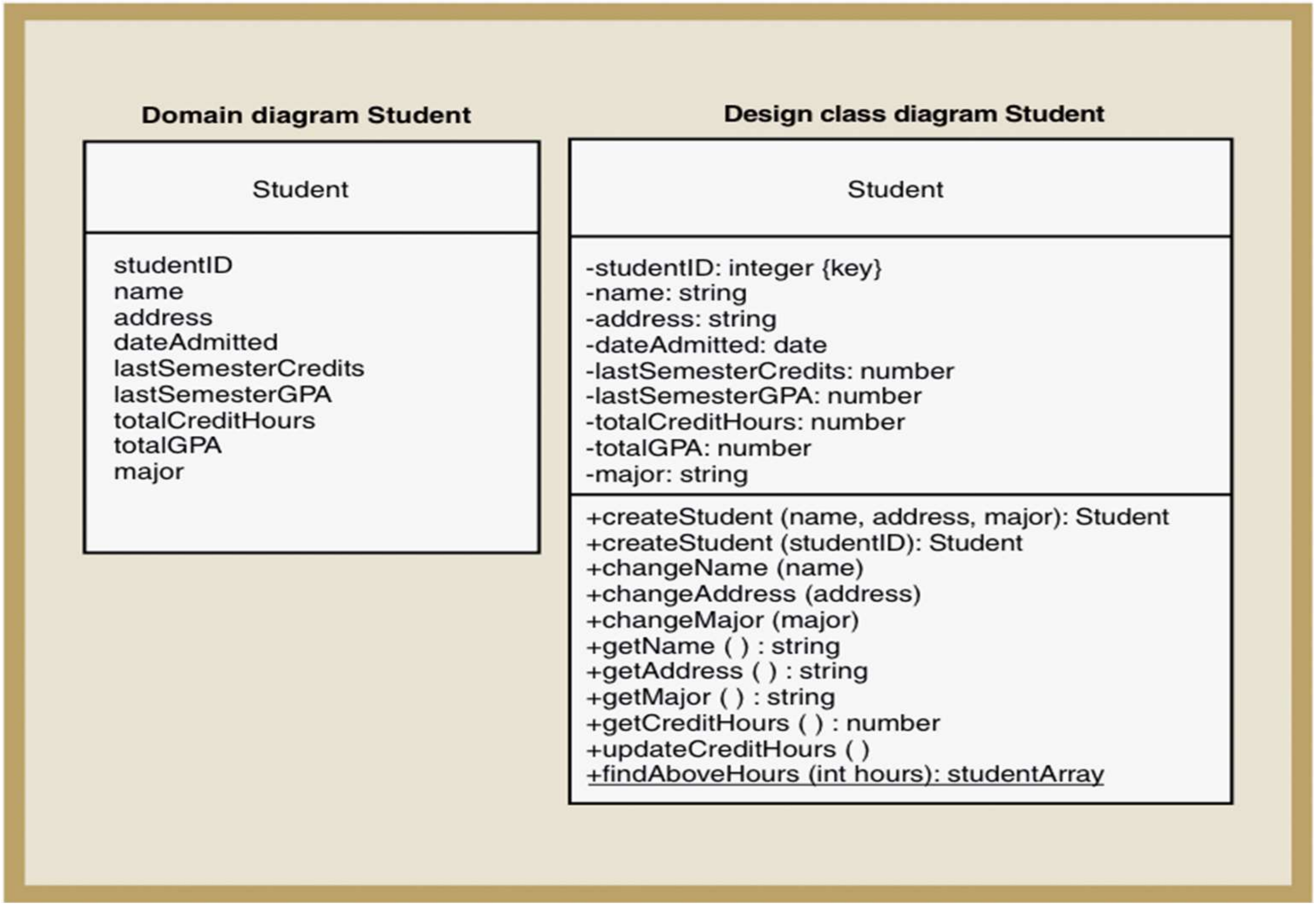

StudentCourseAssociation
- m_Student: Student
- m_Course: Course
- m_Grade: Grade
+ «property» Course() : Course
+ «property» Grade() : Grade
+ «property» Student() : Student
*
*
Student
1
1
Course

Figure 8: Association Class

Association classes often occur in many-to-many associations.

Student Class Examples for the Domain Class and the Design Class Diagrams



Domain diagram Student

Student

studentID
name
address
dateAdmitted
lastSemesterCredits
lastSemesterGPA
totalCreditHours
totalGPA
major

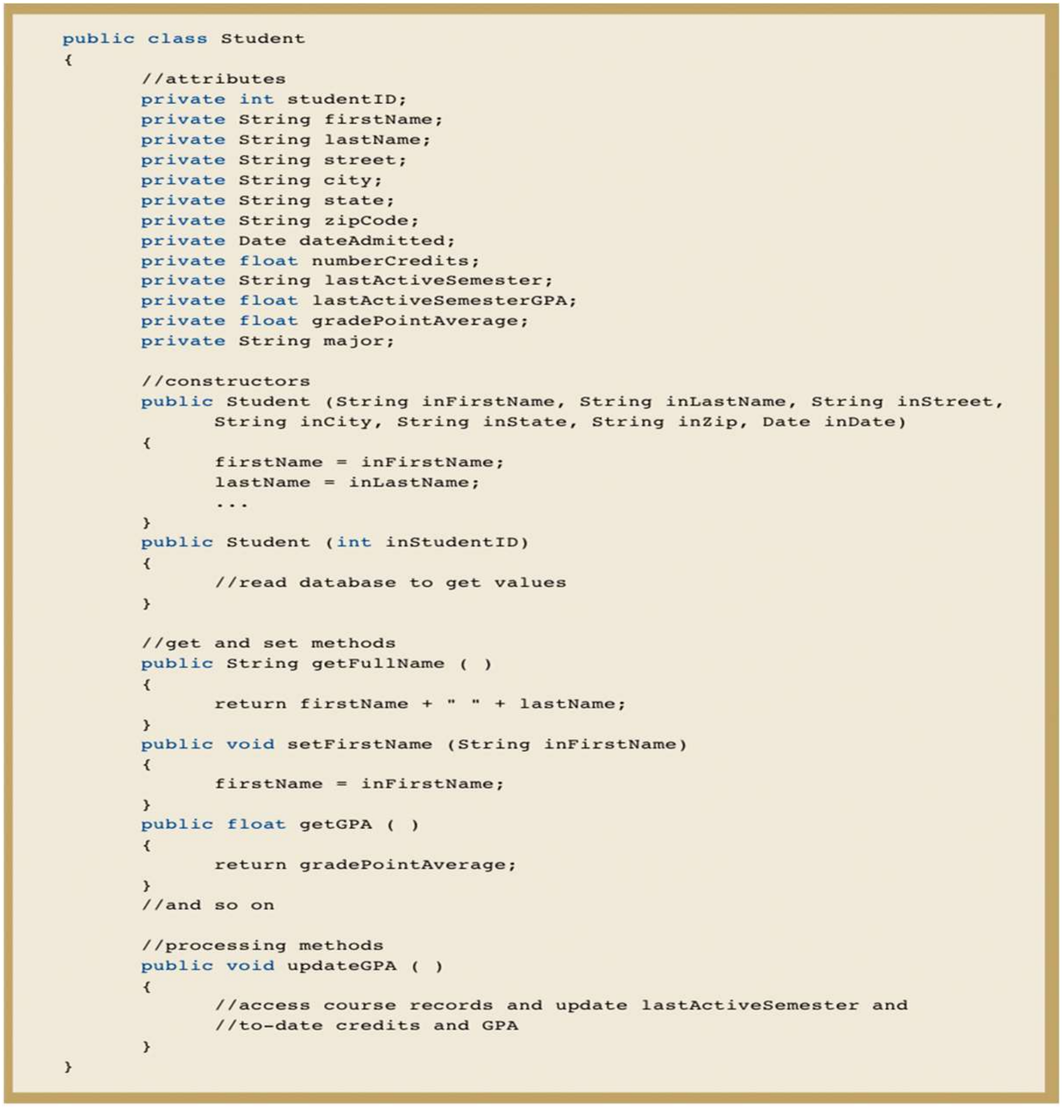
Design class diagram Student

Student

-studentID: integer {key}
-name: string
-address: string
-dateAdmitted: date
-lastSemesterCredits: number
-lastSemesterGPA: number
-totalCreditHours: number
-totalGPA: number
-major: string

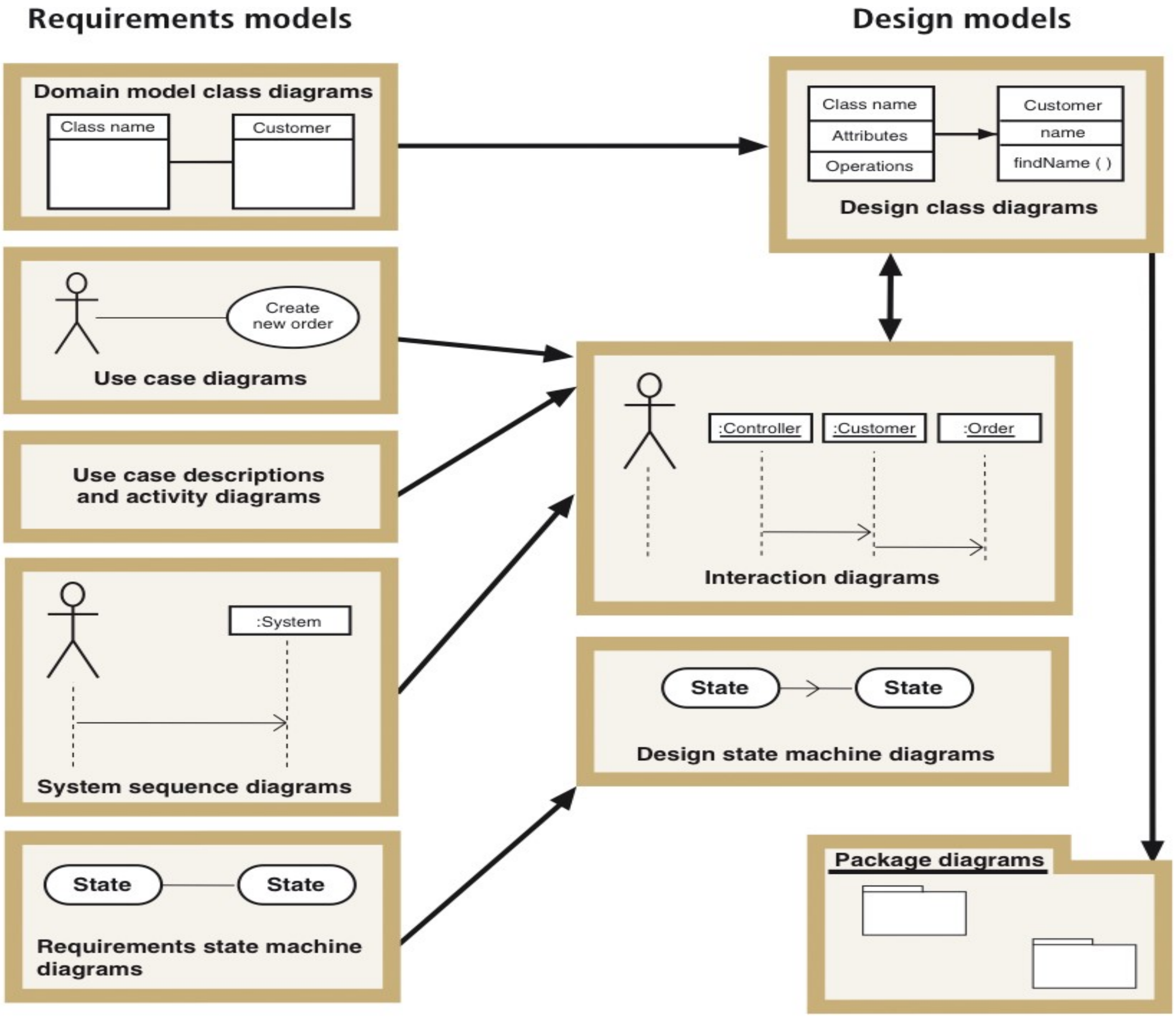
+createStudent (name, address, major): Student
+createStudent (studentID): Student
+changeName (name)
+changeAddress (address)
+changeMajor (major)
+getName ( ) : string
+getAddress ( ) : string
+getMajor ( ) : string
+getCreditHours ( ) : number
+updateCreditHours ( )
+findAboveHours (int hours): studentArray

Example Class Definition in Java for Student Class

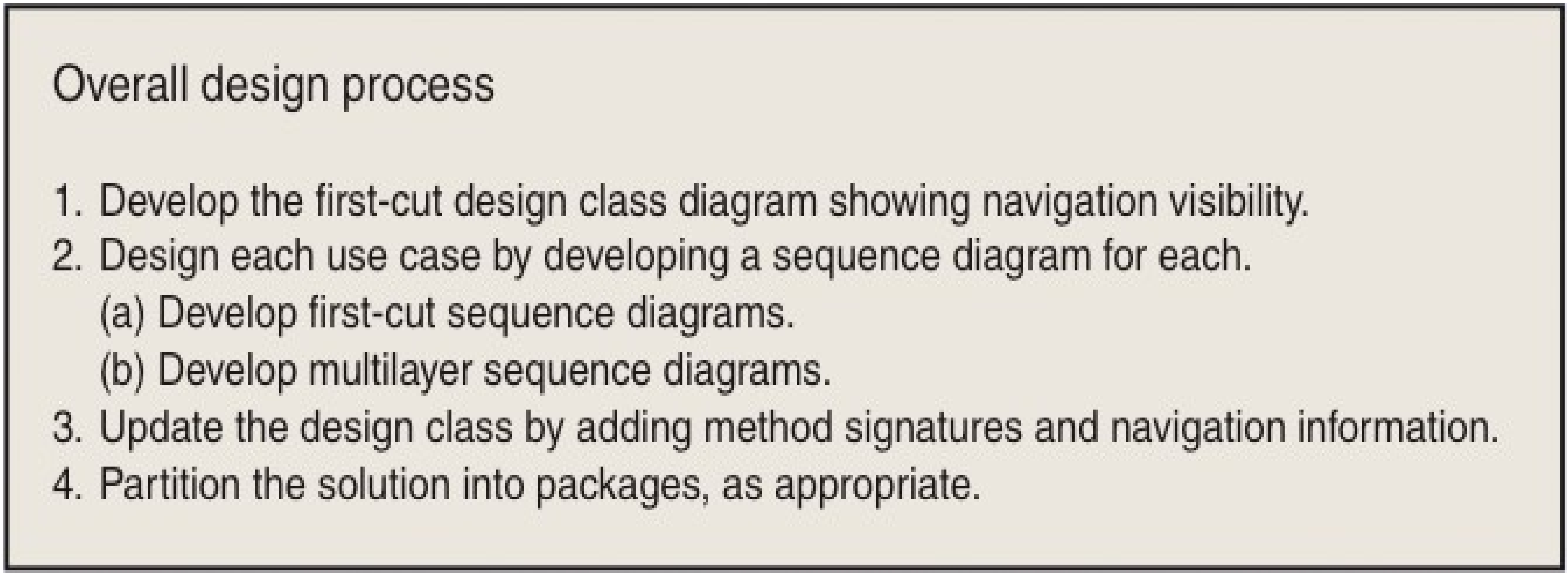


```
public class Student
{
        //attributes
        private int studentID;
        private String firstName;
        private String lastName;
        private String street;
        private String city;
        private String state;
        private String zipCode;
        private Date dateAdmitted;
        private float numberCredits;
        private String lastActiveSemester;
        private float lastActiveSemesterGPA;
        private float gradePointAverage;
        private String major;

        //constructors
        public Student (String inFirstName, String inLastName, String inStreet,
              String inCity, String inState, String inZip, Date inDate)
        {
              firstName = inFirstName;
              lastName = inLastName;
              ...
        }
        public Student (int inStudentID)
        {
              //read database to get values
        }

        //get and set methods
        public String getFullName ( )
        {
              return firstName + " " + lastName;
        }
        public void setFirstName (String inFirstName)
        {
              firstName = inFirstName;
        }
        public float getGPA ( )
        {
              return gradePointAverage;
        }
        //and so on

        //processing methods
        public void updateGPA ( )
        {
              //access course records and update lastActiveSemester and
              //to-date credits and GPA
        }
}
```

Design Models with Their Respective Input Models


Requirements models
Design models
Domain model class diagrams
Class name
Customer
Use case diagrams
Create new order
Use case descriptions and activity diagrams
System sequence diagrams
:System
Requirements state machine diagrams
State
State
Design class diagrams
Class name
Attributes
Operations
Customer
name
findName ( )
Interaction diagrams
:Controller
:Customer
:Order
Design state machine diagrams
State
State
Package diagrams

Realization of use case – specification of all detailed system processing for each use case



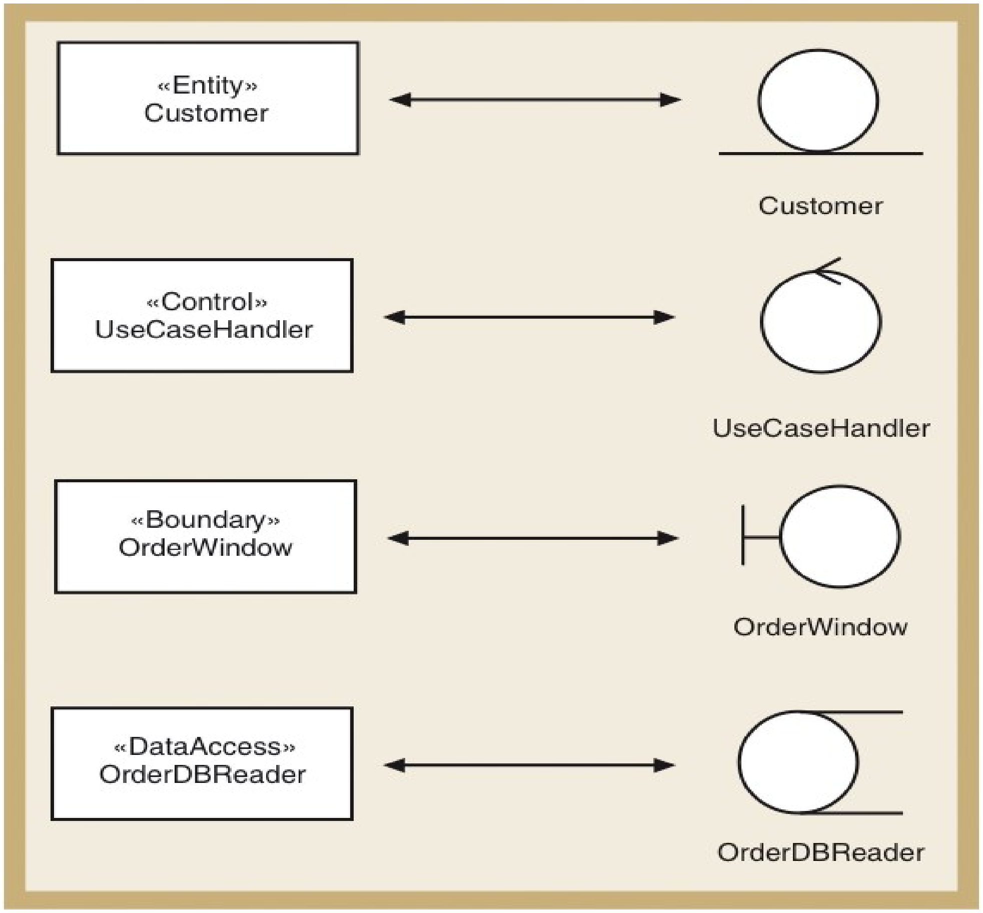
Overall design process

1. Develop the first-cut design class diagram showing navigation visibility.
2. Design each use case by developing a sequence diagram for each.
    (a) Develop first-cut sequence diagrams.
    (b) Develop multilayer sequence diagrams.
3. Update the design class by adding method signatures and navigation information.
4. Partition the solution into packages, as appropriate.

Design Class Symbols

- UML does not distinguish between design class notation and domain model notation
- Domain model class diagram shows conceptual classes in users' work environment
- Design class diagram specifically defines software classes
- UML uses stereotype notation to categorize a model element by its characteristics

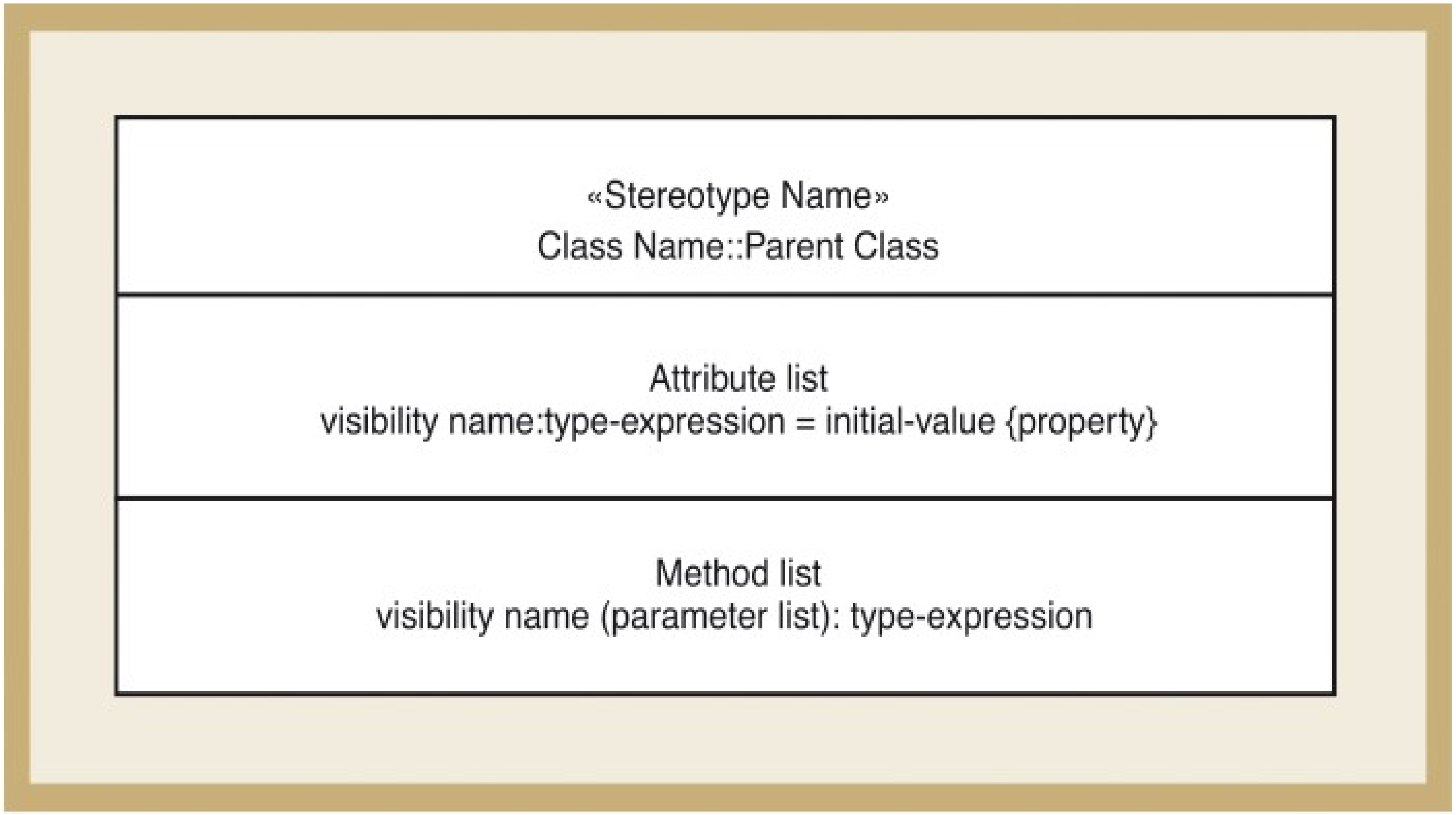
Design Class Symbols


«Entity»
Customer
Customer
«Control»
UseCaseHandler
UseCaseHandler
«Boundary»
OrderWindow
OrderWindow
«DataAccess»
OrderDBReader
OrderDBReader

Standard Design Classes

- Entity – design identifier for problem domain class
  - Persistent class – exists after system is shut down – entities usually persistent
- Control – mediates between boundary and entity classes, between the view layer and domain layer
- Boundary – designed to live on system's automation boundary, touched by users
  - User interface and windows classes
- Data access – retrieves data from and sends data to database
  - Control, boundary and Data access classes – not persistent

Notation Used to Define a Design Class

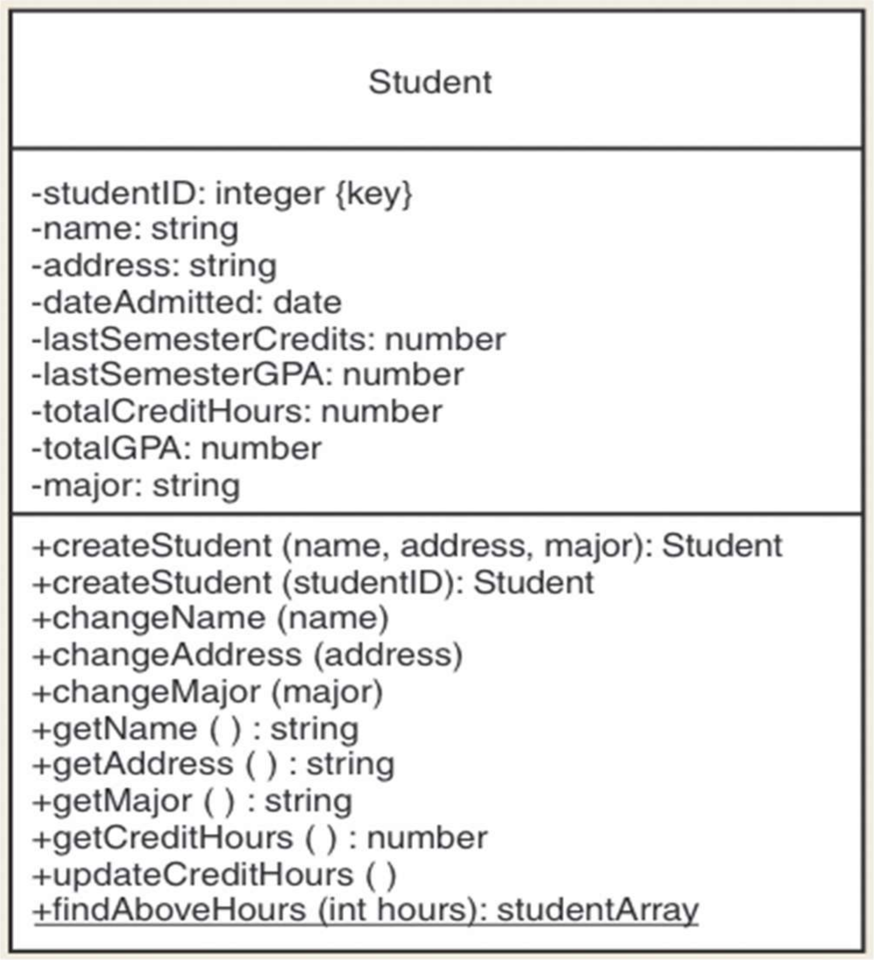


«Stereotype Name»
Class Name::Parent Class

Attribute list
visibility name:type-expression = initial-value {property}

Method list
visibility name (parameter list): type-expression

Student Design Class Example



Student

-studentID: integer {key}
-name: string
-address: string
-dateAdmitted: date
-lastSemesterCredits: number
-lastSemesterGPA: number
-totalCreditHours: number
-totalGPA: number
-major: string

+createStudent (name, address, major): Student
+createStudent (studentID): Student
+changeName (name)
+changeAddress (address)
+changeMajor (major)
+getName ( ) : string
+getAddress ( ) : string
+getMajor ( ) : string
+getCreditHours ( ) : number
+updateCreditHours ( )
+findAboveHours (int hours): studentArray

Composition

Sometimes a class is composed of other classes.

Composition associations form a whole-part relationship.

In this relationship, the life of the part depends on the whole.

In other words, when the whole is destroyed or removed, the part is also.

Moreover, the part cannot belong to more than one whole.

This implies that the cardinality of the whole to the part is always “1”.

Composition

In a composition association, the whole manages the lifecycle of the part.

To identify composition associations, look for classes that can not exist without other classes.
Look for "wholes" and "parts".

Ask, "Is this class part of another class?"

For example, a Sentence class is part of a Paragraph class that is part of a Chapter class that is part of a Book class.

"Does this class depend on another class?"

The key question to ask, "Is this class destroyed when another class is destroyed?"

Aggregation

- Aggregation is a weaker form of a composition.

- The key difference between an aggregation and a composition is, in an aggregation, the part is not destroyed when the whole is destroyed.

- In an aggregation relationship, the part may be independent of the whole but the whole requires the part.

Aggregation

- An automobile, for example, is composed of an engine, chassis, wheels, etc..

- The wheels are required for the definition for an automobile, but, they are independent and not necessarily destroyed when the automobile is destroyed.

- Aggregation is often referred to as a "Has-a" relationship, as in, an Automobile Has-a(n) engine.

- To identify aggregation associations, ask, "Is this class part of another class and is it independent of the other class?“

- To avoid confusion, when the decision to use the composition/aggregation association is unclear, it's best to model the relationship as a simple association.

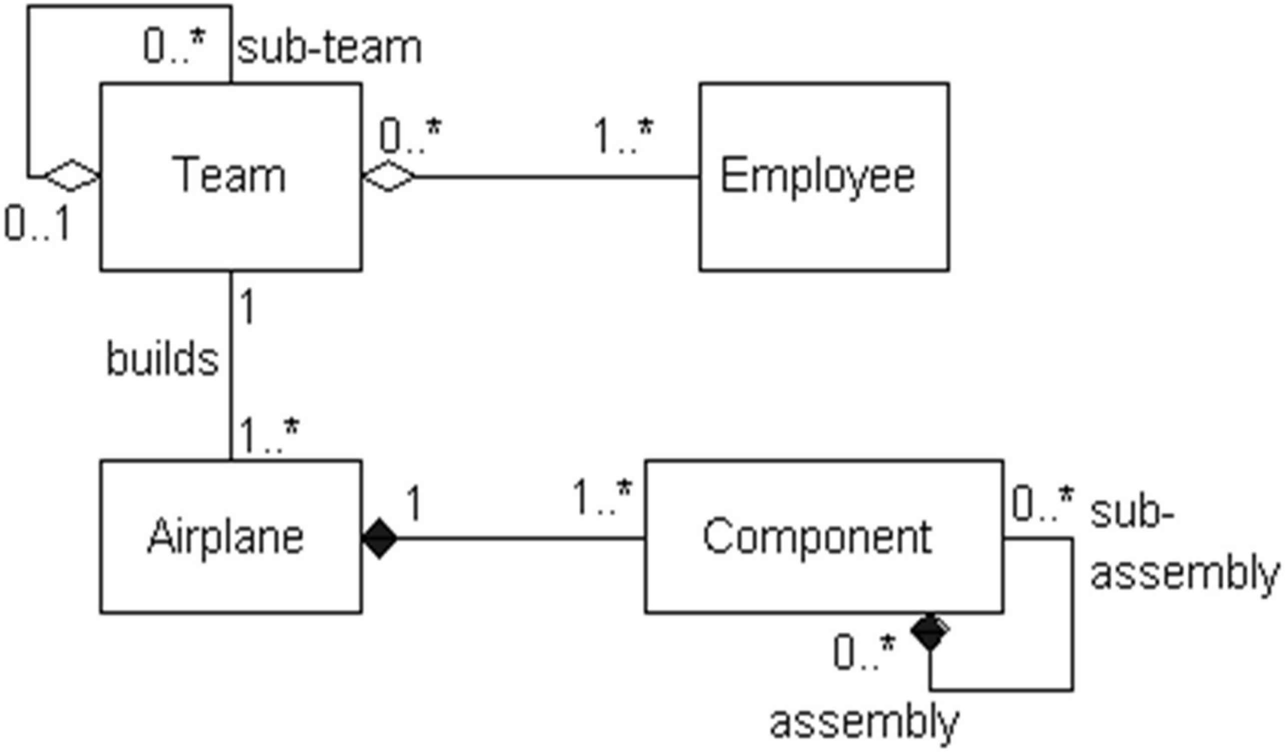
Aggregation and Composition Guidelines

- Sometimes an object is made up of other objects.
- For example, an airplane is made up of a fuselage, wings, engines, landing gear, flaps, and so on.
- A delivery shipment contains one or more packages.
- A team consists of two or more employees.

Aggregation and Composition Guidelines

- These are all examples of the concept of aggregation, which represents "is part of" relationships.
- An engine is part of a plane, a package is part of a shipment, and an employee is part of a team.
- Aggregation is a specialization of association, specifying a whole-part relationship between two objects.
- Composition is a stronger form of aggregation where the whole and parts have coincident lifetimes, and it is very common for the whole to manage the lifecycle of its parts.
- From a stylistic point of view, because aggregation and composition are both specializations of association the guidelines for associations apply.

Examples of aggregation and composition


0..*
sub-team
Team
0..1
0..*
1..*
Employee
1
builds
1..*
Airplane
1
1..*
Component
0..*
sub-
assembly
0..*
assembly

1. Apply the Sentence Rule for Aggregation
2. You Should Be Interested In Both The Whole And The Part
3. Depict the Whole to the Left of the Part
4. Apply Composition to Aggregates of Physical Items
5. Apply Composition When the Parts Share The Persistence Lifecycle With the Whole
6. Don't Worry About Getting the Diamonds Right

Generalization/Specialization

- The generalization/specialization association exists when one class is a specialized version of another class.

- In this relationship, a common, or base, class forms the foundation for more specialized, or derived, classes.

- This association is commonly referred to as inheritance because the derived classes inherit the functionality of their base classes to provide specialized behavior.

Inheritance

- Inheritance provides the mechanism for new classes to be formed from the attributes and behavior of existing classes

- As base classes are specialized into derived classes, hierarchies are formed that represent *is-a* or *kind-of* relationships.

- For example, an Algebra course *is-a* specialization of a Math course; a student is a *kind-of* Person.

Inheritance

- To identify inheritance relationships, ask, "Is this class a specialization of a more general class?“

- Beware of the tendency to see all associations in terms of inheritance.

- This leads to complex and deep class hierarchies that are difficult to develop.

- As a rule of thumb, limit hierarchies to six or fewer levels.

Inheritance

- Inheritance is best suited for relatively shallow class hierarchies.
- Use Inheritance when:
- The inheritance hierarchy represents an *is-a* relationship and not a *has-a* (composition/aggregate) relationship.
- The same behavior is applied to different data types.
- The class hierarchy is reasonably shallow, and unlikely to become much deeper over time.

Interface inheritance

Closely related to inheritance is the concept of interface inheritance.

Like an inherited class, an interface provides a common specification for behavior, but, unlike an inherited class, an interface cannot be created.

An interface simply specifies common behavior that other classes inherit but implement differently.

This implies that unrelated classes can provide independent implementations for the same behavior specification (polymorphism).

Interface inheritance

For example, the .NET framework provides the IComparable interface that defines a generalized comparison method.

This interface is typically used for sorting purposes.

Classes that inherit the IComparable interface implement behavior specific to their needs.

So, using the same IComparable interface, an Integer class can sort integers, a Byte class can sort bytes, a Student class can sort students, and so on.

Interface inheritance

- The behavior has been abstracted out from any particular class and specifically implemented by all classes that need it.
- Use interface inheritance when:
- Unrelated object types need to provide the same behavior.
- Multiple inheritance is needed (very difficult to implement using the inheritance association).
- Inheritance is prohibited. For example, C# structures cannot inherit from classes, but they can implement interfaces.
- The Strategy Design Pattern is an excellent candidate for an interface inheritance.

Inheritance Guidelines

- Inheritance models "is a" and "is like" relationships, enabling you to easily reuse existing data and code.

- When "A" inherits from "B" we say that "A" is the subclass of "B" and that "B" is the superclass of "A.

- "Furthermore, we say that we have "pure inheritance" when "A" inherits all of the attributes and methods of "B."

- The UML modeling notation for inheritance is a line with a closed arrowhead pointing from the subclass to the superclass.

Inheritance Guidelines

1. Apply the Sentence Rule For Inheritance
2. Place Subclasses Below Superclasses
3. Beware of Data-Based Inheritance
4. A Subclass Should Inherit Everything

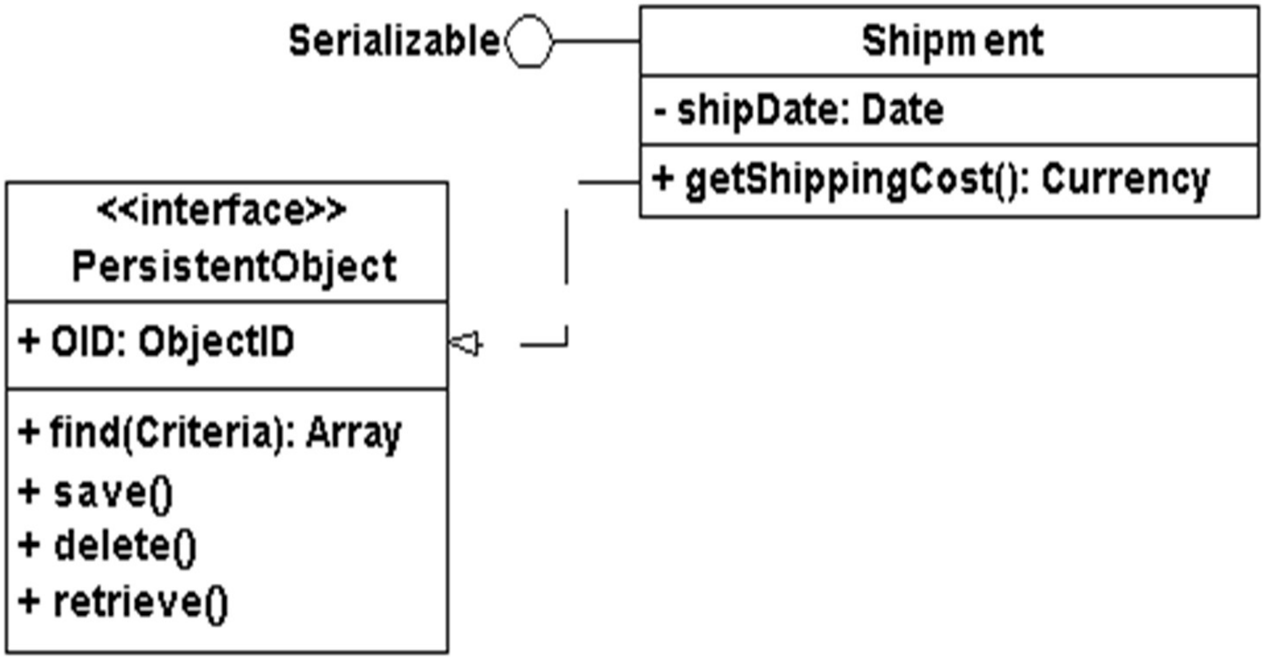
Interfaces

- An interface is a collection of operation signature and/or attribute definitions that ideally defines a cohesive set of behaviors.

- Interfaces are implemented, "realized" in UML parlance, by classes and components - to realize an interface a class or component must implement the operations and attributes defined by the interface.

- Any given class or component may implement zero or more interfaces and one or more classes or components can implement the same interface.

Interfaces on UML class diagrams.



Serializable
Shipment
- shipDate: Date
+ getShippingCost(): Currency
<<interface>>
PersistentObject
+ OID: ObjectID
+ find(Criteria): Array
+ save()
+ delete()
+ retrieve()

1.Interface Definitions Must Reflect Implementation Language Constraints.
In Figure you see that a standard class box has been used to define the interface *PersistentObject* (note the use of the <> stereotype).

2.Name Interfaces According To Language Naming Conventions

3.Apply "Lollipop" Notation To Indicate That A Class Realizes an Interface

4.Define Interfaces Separately From Your Classes

5.Do Not Model the Operations and Attributes of an Interface in Your Classes. In Figure you'll notice that the *Shipment* class does not include the attributes or operations defined by the two interfaces that it realizes.

6.Consider an Interface to Be a Contract

1. Conceptual Class Diagrams

- Let us start at a simple UML class diagram for the conceptual model for a university.

- Classes are depicted as boxes with three sections, the top one indicates the name of the class, the middle one lists the attributes of the class, and the third one lists the methods.

- By including both an attribute and a method box in the class we are arguably making design decisions in our model, something we shouldn't be doing if our goal is conceptual modeling.

- Another approach would be to have two sections, one for the name and one listing responsibilities.

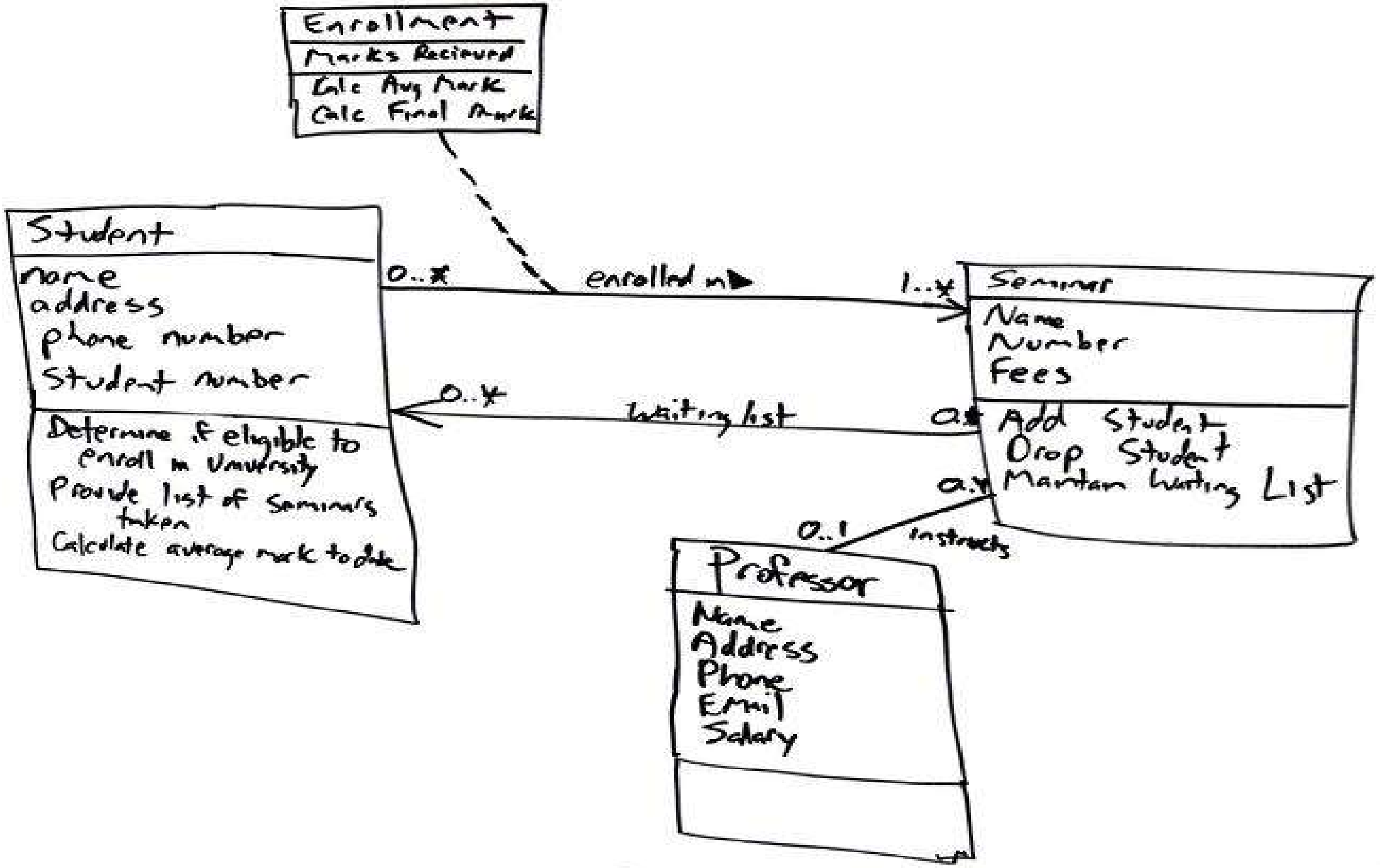
- This would be closer to a CRC model (so if we wanted to take this sort of approach we'd use CRC cards instead of a UML class diagram).

- we could also use class boxes that show just the name of the class, enabling me to focus on just the classes and their relationships.

- However, if that was our goal we'd be more likely to create an ORM diagram instead.

- In short, we prefer to follow AM's *Apply the Right Artifact(s)* practice and use each modeling technique for what it's best at.

Figure 1. Sketch of a conceptual class diagram.


Enrollment
Marks Recieved
Calc Avg Mark
Calc Final Mark
Student
name
address
phone number
Student number
Determine if eligible to enroll in University
Provide list of Seminars taken
Calculate average mark to date
0..*
enrolled in ▶
1..*
Seminar
Name
Number
Fees
Add Student
Drop Student
Maintain Waiting List
0..*
waiting list
0..*
0..*
0..1
instructs
Professor
Name
Address
Phone
Email
Salary

- *Enrollment* is an associative class, also called a link class, which is used to model associations that have methods and attributes.

- Associative classes are typically modeled during analysis and then refactored into what we show in Figure 2 during design (Figure 2 is still a conceptual diagram, albeit one with a design flavor to it).

- No mainstream programming language exists that supports the notion of associations that have responsibilities.

- Because we can directly build your software in this manner, we have a tendency to stay away from using association classes and instead resolve them during my analysis efforts.

- This is not a purist way to model, but it is pragmatic because the other members on the team, including stakeholders, don't need to learn the notation and concepts behind associative classes.

- Figure 2 depicts a reworked version of Figure 1, the associative class has been resolved.

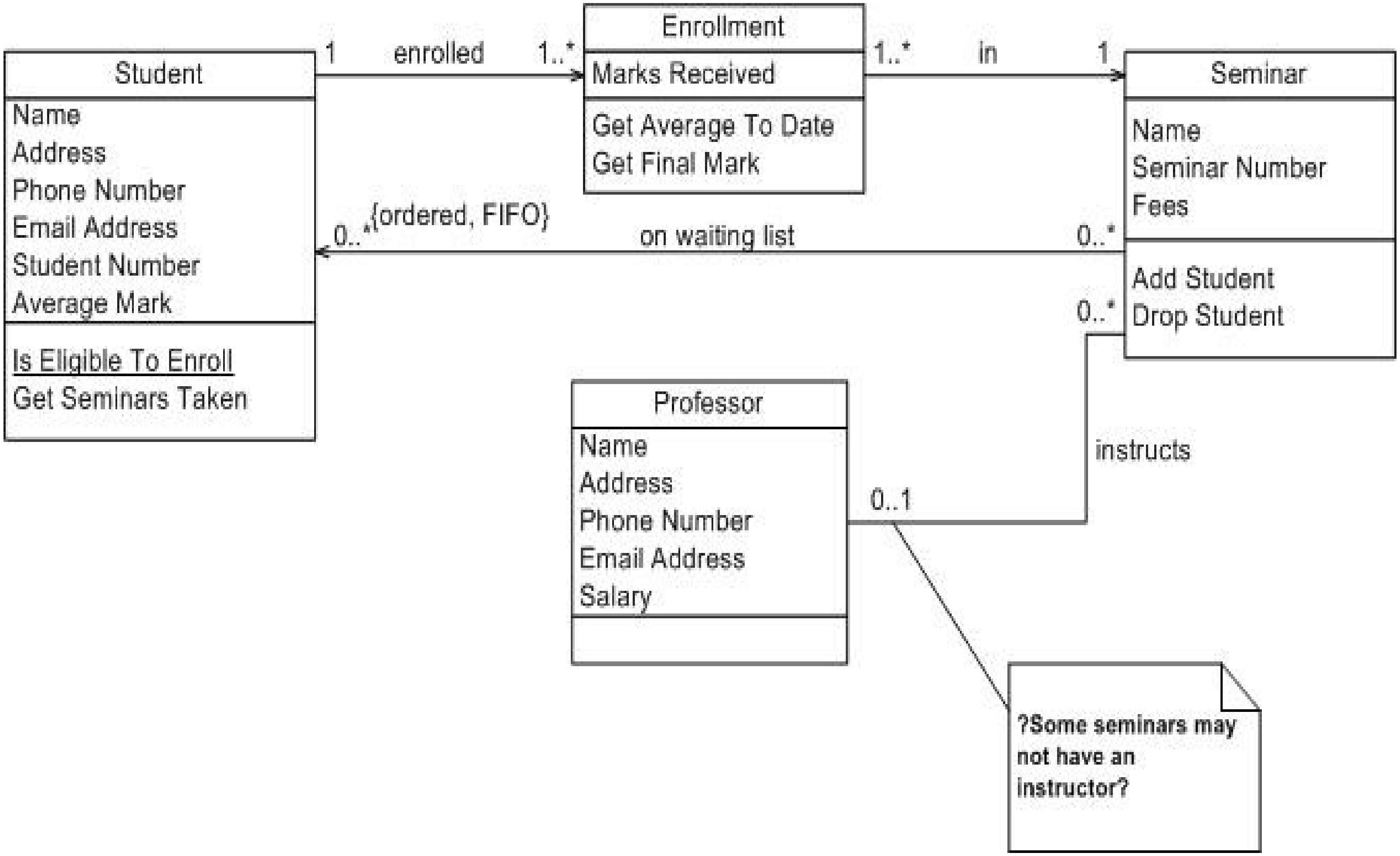
- We could have added an attribute in the *Seminar* class called *Waiting List* but, instead, chose to model it as an association because that is what it actually represents: that seminar objects maintain a waiting list of zero or more student objects.

- Attributes and associations are both properties in the UML 2.0 so they're treated as basically the same sort of thing.

- We also showed associations are implemented as a combination of attributes and operations - We prefer to keep the models simple and assume that the attributes and operations exist to implement the associations.

- Furthermore that would be a detailed design issue anyway, something that isn't appropriate on a conceptual model.

Figure 2. Initial conceptual class diagram.


Student
Name
Address
Phone Number
Email Address
Student Number
Average Mark
Is Eligible To Enroll
Get Seminars Taken
1
enrolled
1..*
Enrollment
Marks Received
Get Average To Date
Get Final Mark
1..*
in
1
Seminar
Name
Seminar Number
Fees
Add Student
Drop Student
0..*
{ordered, FIFO}
on waiting list
0..*
0..*
instructs
0..1
Professor
Name
Address
Phone Number
Email Address
Salary
?Some seminars may not have an instructor?

- The *on waiting list* association is unidirectional because there isn't yet a need for collaboration in both directions.

- Follow the AM practice of *Create Simple Content* and don't over model - you don't need a bi-directional association right now so don't model it.

- The *enrolled in* association between the *Student* and *Enrollment* classes is also uni-directional for similar reasons.

- For this association it appears student objects know what enrollment records they are involved with, recording the seminars they have taken in the past, as well as the seminars in which they are currently involved.

- This association would be traversed to calculate their student object's average mark and to provide information about seminars taken.

- There is also an *enrolled in* association between *Enrollment* and *Seminar* to support the capability for student objects to produce a list of seminars taken.

- The *instructs* association between the *Professor* class and the *Seminar* class is bidirectional because professor objects know what seminars they instruct and seminar objects know who instruct them.

- When we do conceptual modeling the style is to name attributes and methods using the formats *Attribute Name* and *Method Name*, respectively.
- Following a consistent and sensible naming convention helps to make your diagrams readable, an important benefit of AM's *Apply Modeling Standards* practice.
- Also notice in Figure 2 how we haven't modeled the visibility of the attributes and methods to any great extent.
- Visibility is an important issue during design but, for now, it can be ignored.
- Also notice we haven't defined the full method signatures for the classes.
- This is another task we typically leave to design.

- We were able to determine with certainty, based on this information, the multiplicities for all but one association and for that one we marked it with a note.

- so we, know to discuss it further with my stakeholders.

- Notice the use of question marks in the note.

- Our style is to mark unknown information on my diagrams this way to remind myself that we need to look into it.

- In Figure 2 we modeled a UML constraint, in this case *{ordered FIFO}* on the association between *Seminar* and *Student*.

- The basic idea is that students are put on the waiting list on a first-come, first-served/out (FIFO) basis.

- In other words, the students are put on the waiting list in order.

- UML constraints are used to model complex and/or important information accurately in your UML diagrams.

- UML constraints are modeled using the format “{constraint description}” format, where the constraint description may be in any format, including predicate calculus.

- Our preference is to use UML notes with English comments, instead of formal constraints, because they're easier to read.

Conclusion

- Sometimes, developers don't know where or how to begin identifying classes.
- Often classes are simply chosen based on what seems to make sense to the developer without regarding class associations or cohesiveness.
- As a result, the logical structure for an application can contain classes that are unnecessary and complex making the application hard to extend and maintain.
- Todays session provided tips and guidance to identify object-oriented, cohesive classes that work together to accomplish the application's business function.

BITS Pilani
Pilani Campus

SOFTWARE DESIGN PRINCIPLES

Unified Modeling Language for Design

By Prof A R Rahman

Software class diagram

Class diagram

- A class diagram is a diagram used in designing and modeling software to describe classes and their relationships.

- Class diagrams enable us to model software in a high level of abstraction and without having to look at the source code.

- Classes in a class diagram correspond with classes in the source code.

- The diagram shows the names and attributes of the classes, connections between the classes, and sometimes also the methods of the classes.

Class diagram

- A class diagram depicts classes and their interrelationships
- Used for describing structure and behavior in the use cases
- Provide a conceptual model of the system in terms of entities and their relationships
- Used for requirement capture, end-user interaction
- Detailed class diagrams are used by developers

Class diagram

- Each class is represented by a rectangle subdivided into three compartments
  - Name
  - Attributes
  - Operations

- Modifiers are used to indicate visibility of attributes and operations.
  - ‘+’ is used to denote *Public* visibility (everyone)
  - ‘#’ is used to denote *Protected* visibility (friends and derived)
  - ‘-’ is used to denote *Private* visibility (no one)

- By default, attributes are hidden and operations are visible.

Member access modifiers

- All classes have different access levels depending on the access modifier (visibility).
- Here are the access levels with their corresponding symbols:
- Public (+)
- Private (-)
- Protected (#)
- Package (~)
- Derived (/)
- Static (underlined)

Class diagram

- When it comes to system construction, a class diagram is the most widely used diagram.

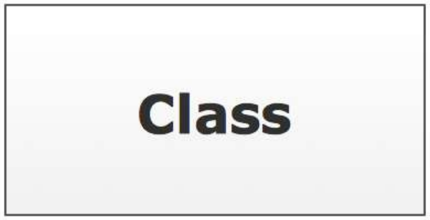
- UML Class Diagrams is a type of static structure diagram that is used for general conceptual modeling of the systematics of the application.

- Such a diagram would illustrate the object-oriented view of a system .

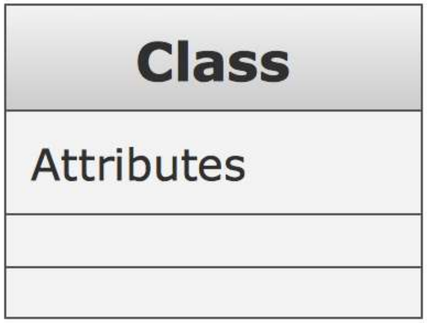
- The object orientation of a system is indicated by a class diagram.

- It describes the structure of a system by showing the:

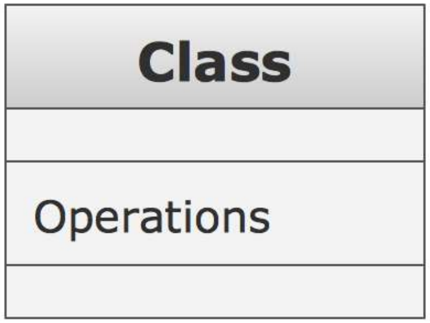
- classes of a system,
- attributes,
- objects,
- operations, and the relationships between classes.



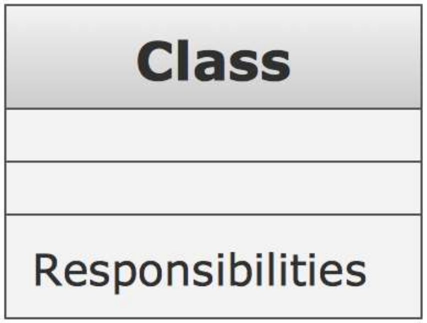
Class represents a set of objects that have the same structure, behavior, and relationships with objects of other classes.


Class
Attributes

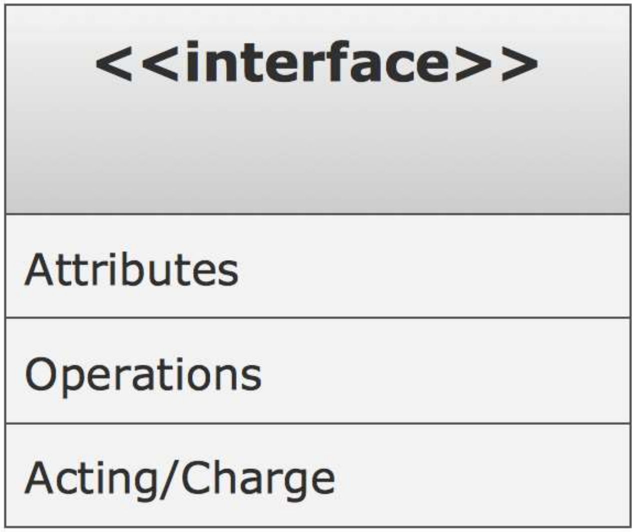
Attribute is a typed value that defines the properties and behavior of the object.


Class
Operations

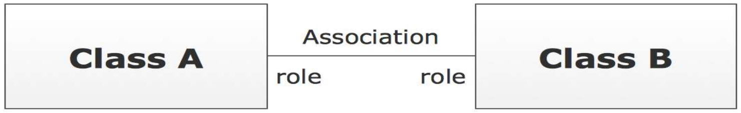
Operation is a function that can be applied to the objects of a given class.


Class
Responsibilities

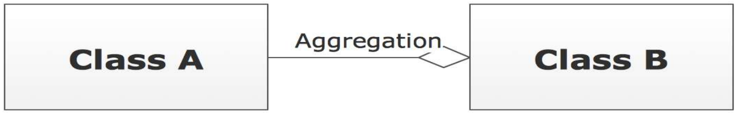
Responsibility is a contract which the class must conform.


<<interface>>
Attributes
Operations
Acting/Charge

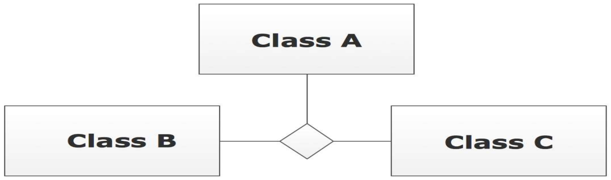
Interface is an abstract class that defines a set of operations that the object of the class associated with this interface provides to other objects.


Class A
Association
role
role
Class B

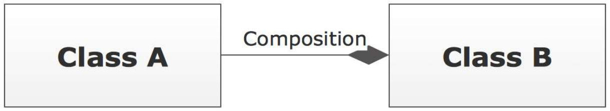
Association is a relationship that connect two classes.


Class A
Aggregation
Class B

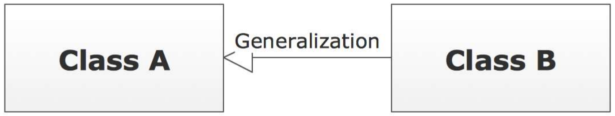
Aggregation is an association with the relation between the whole and its parts, the relation when one class is a certain entity that includes the other entities as components.


Class A
Class B
Class C

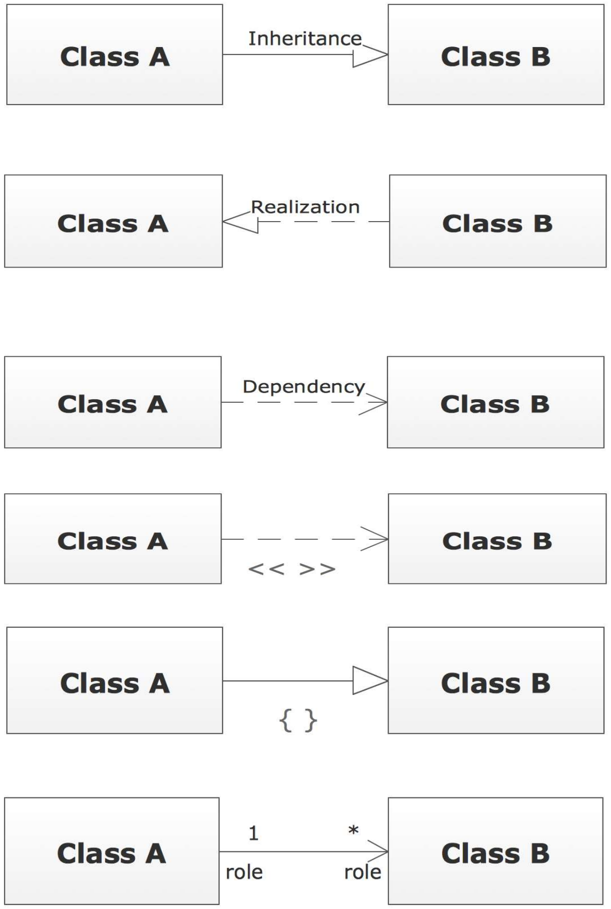
N-ary association represents two or more aggregations.


Class A
Composition
Class B

Composition is a strong variant of aggregation when parts cannot be separately of the whole.


Class A
Generalization
Class B

Generalization is an association between the more general classifier and the more special classifier.


Class A
Inheritance
Class B
Class A
Realization
Class B
Class A
Dependency
Class B
Class A
<< >>
Class B
Class A
{ }
Class B
Class A
1
*
role
role
Class B

Inheritance is a relationship when a child object or class assumes all properties of his parent object or class.

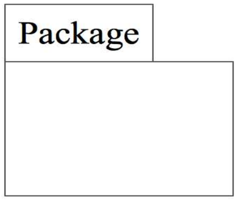
Realization is a relationship between interfaces and classes or components that realize them.

Dependency is a relationship when some changes of one element of the model can need the change of another dependent element.

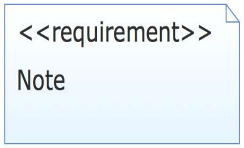
Allows to define the properties of the dependency relationship between classes or classes and packages.

Allows to indicate the additional properties of association.

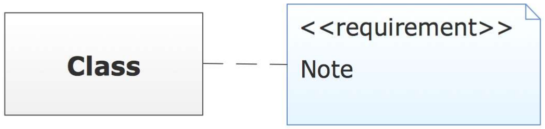
Multiplicity shows the quantity of instances of one class that are linked to one instance of the other class.



Package groups the classes and other packages.

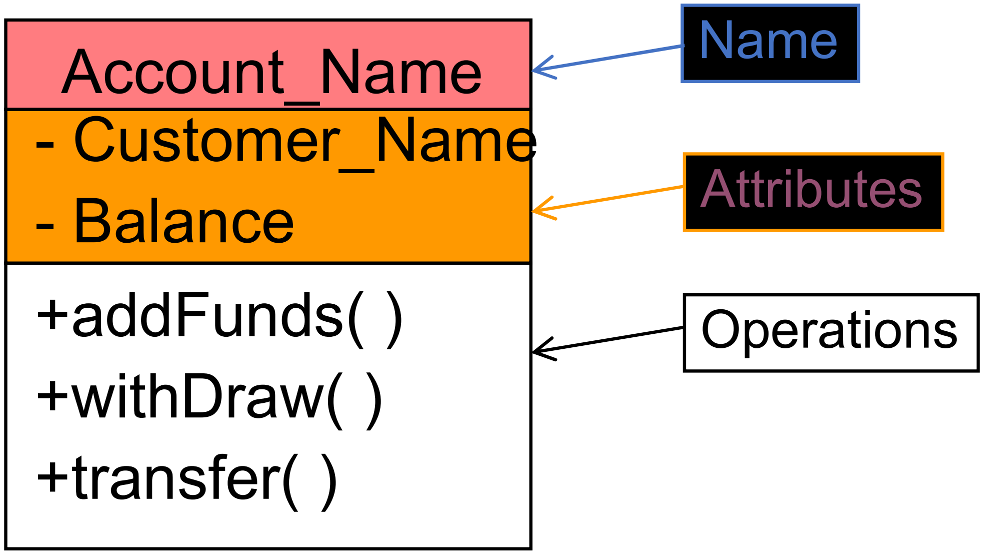


Note is a textual explication.



Note connector is a connection between the note and elements.

Class diagram


Account_Name
- Customer_Name
- Balance
+addFunds( )
+withDraw( )
+transfer( )
Name
Attributes
Operations

- Class attributes

- Class attributes are represented by a rectangle with two tabs that indicates a software element.

- Links

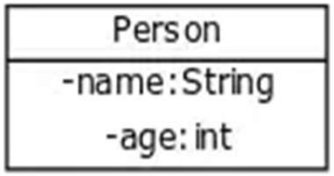
- Links are the lines that connect two shapes of an object diagram to each other.

Describing class and class attributes

- First we will describe one class and its attributes.
- public class Person {
- private String name;
- private int age;
- }

Describing class and class attributes

- In a class diagram, a class is represented by a rectangle with the name of the class written on top.
- A line below the name of the class divides the name from the list of attributes (names and types of the class variables).
- The attributes are written one attribute per line.
- In a class diagram, class attributes are written "attributeName: attributeType".
- A + before the attribute name means the attribute is public, and a - means the attribute is private.



Describing class constructor

- Below we have the source code for a constructor for our Person class.

- The constructor gets the name of the person as a parameter.

```
public class Person {
   private String name;
   private int age;

 public Person(String initialName) {
   this.name = initialName;
   this.age = 0;
   }
 }
```

Describing class constructor

- In a class diagram, we list the constructor (and all other methods) below the attributes.

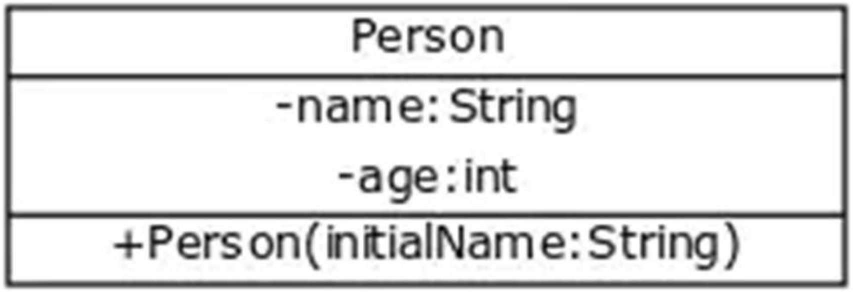
- A line below the attributes list separates it from the method list.

- Methods are written with +/- (depending on the visibility of the method),

- method name, parameters, and their types.

- The constructor above is written +Person(initialName:String)

- The parameters are written the same way class attributes are — "parameterName: parameterType".



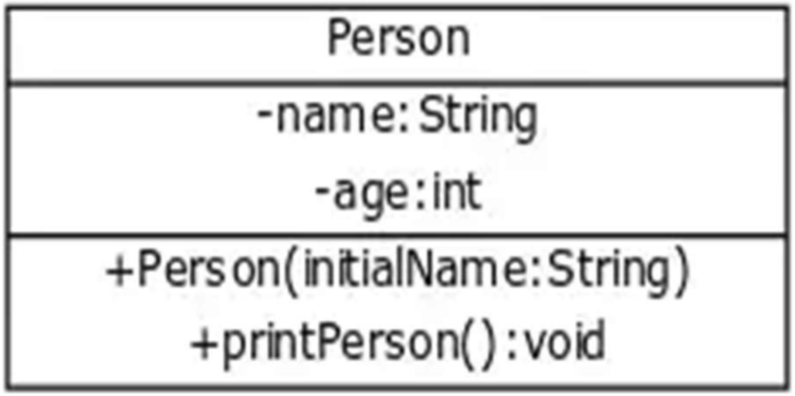
Describing class methods

- Below we have added a method printPerson() which returns void to the Person class.

```
public class Person {
    private String name;
    private int age;

    public Person(String initialName) {
        this.name = initialName;
        this.age = 0;
    }

    public void printPerson() {
        System.out.println(this.name + ", age " +   this.age + " years");
    }
}
```

Describing class methods

- In a class diagram, we list all class methods including the constructors;
- constructors are listed first and then all class methods.
- We also write the return type of a method in the class diagram.

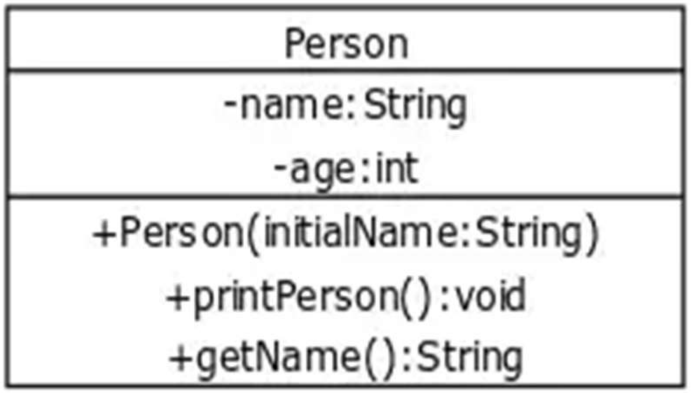

Person
-name: String
-age: int
+Person(initialName: String)
+printPerson() : void

A class diagram describes classes, constructors and methods

- A class diagram describes classes and their attributes, constructors and methods
- as well as the connections between classes.
- However a class diagram tells us nothing about the implementation of the constructors or the methods.
- Therefore a class diagram describes the structure of an object but not its functionality.
- For example the method printPerson uses the class attributes name and age,
- but this cannot be seen from the class diagram.
- We have added method getName to the Person class which returns the name of the Person.

A class diagram describes classes, constructors and methods

```
• public class Person {
•     private String name;
•     private int age;

•     public Person(String initialName) {
•         this.name = initialName;
•         this.age = 0;
•     }

•     public void printPerson() {
•         System.out.println(this.name + ", age " +   this.age + " years");
•     }

•     public String getName() {
•         return this.name;
•     }
• }
```


Person
-name: String
-age:int
+Person(initialName:String)
+printPerson():void
+getName():String

Connections between classes

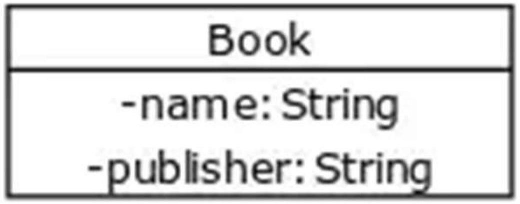
- In a class diagram, the connections between classes are shown as arrows.

- The arrows also show the direction of the connection.

- Below we have a class Book.

- public class Book {
-     private String name;
-     private String publisher;

-     // constructors and methods
- }



Connections between classes

- If we add a variable author, type of which is Person, in the source code

- the variable is declared like all other class variables.

- public class Book {
-     private String name;
-     private String publisher;
-     private Person author;

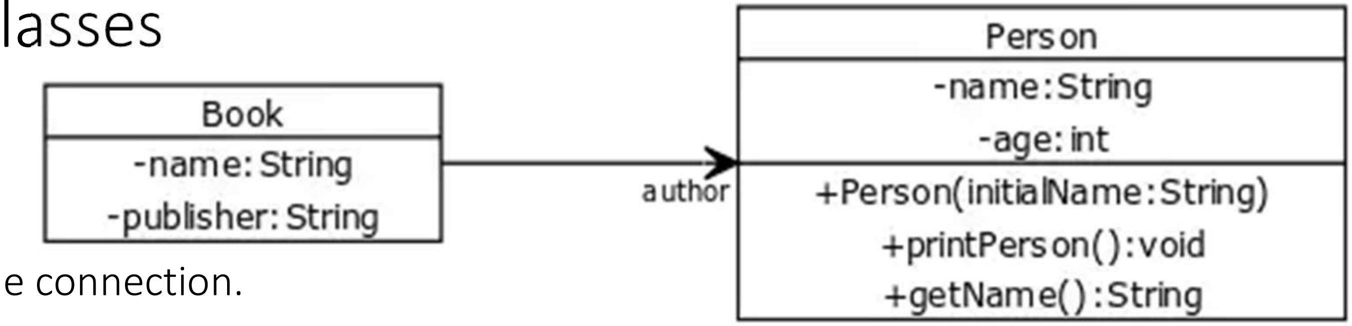
-     // constructors and methods
- }

- In a class diagram variables which refer to other objects are not written with the rest of the class attributes,

- but are shown as connections between the classes.

- In the class diagram below we have the classes Person and Book, and the connection between them.

Connections between classes



Book

-name: String

-publisher: String

author

Person

-name:String

-age: int

+Person(initialName:String)

+printPerson():void

+getName():String

- The arrow shows the direction of the connection.
- The connection above shows that a Book knows its author but a Person
- does not know about books they are the author of.
- We can also add a label to the arrow to describe the connection.
- In the above diagram the arrow has an accompanying label telling us that a Book has an author.
- If a book has multiple authors, the authors are saved to a list.

Connections between classes

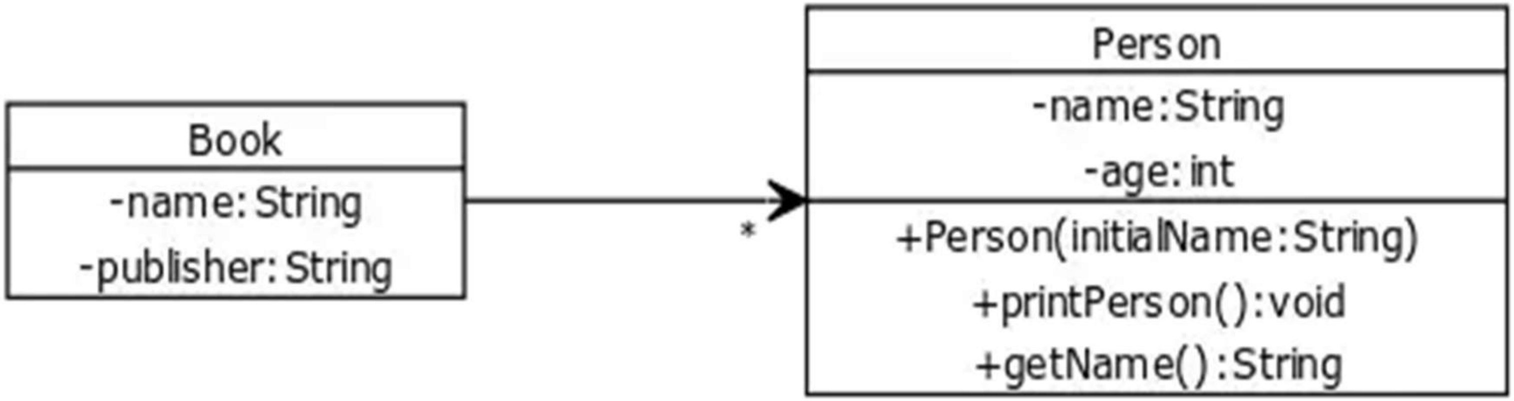
- If a book has multiple authors, the authors are saved to a list.

- public class Book {
-     private String name;
-     private String publisher;
-     private ArrayList<Person> authors;

-     // constructors and methods
- }

- In a class diagram, this situation is described by adding a star to the end of the arrow showing the connection between the classes.
- The star tells us that a book can have between 0 and unlimited number of authors.
- Below we have not amended the label to describe the multiplicity of the connection, but it would be a good idea for the sake of clarity.

Connections between classes



Book
-name: String
-publisher: String
*
Person
-name:String
-age: int
+Person(initialName:String)
+printPerson():void
+getName():String

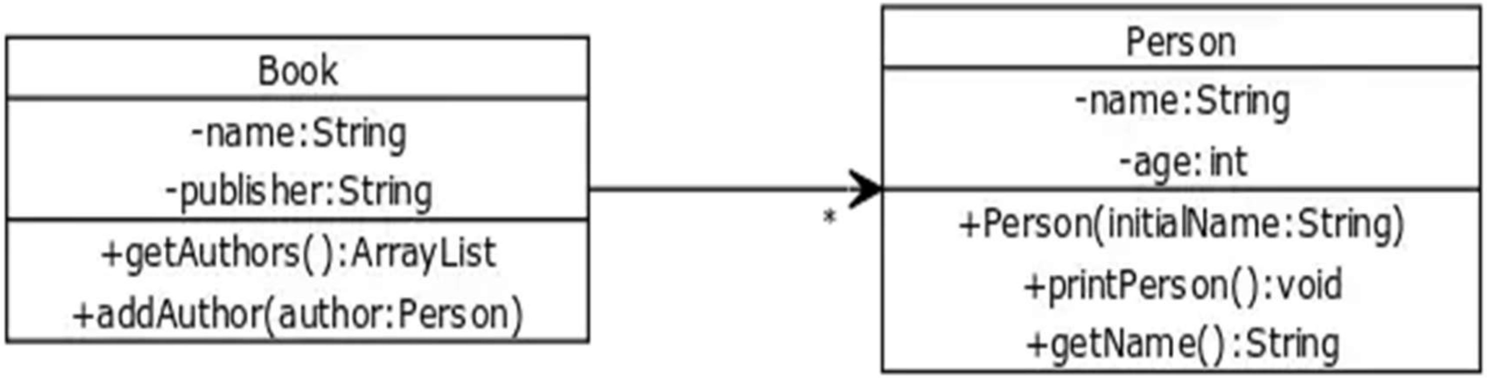
Connections between classes

- Class methods are described just like we did before. Below we have added methods getAuthors and addAuthor to the Book class.

```
• public class Book {
•     private String name;
•     private String publisher;
•     private ArrayList<Person> authors;

•     // constructor

•     public ArrayList<Person> getAuthors() {
•         return this.authors;
•     }

•     public void addAuthor(Person author) {
•         this.authors.add(author);
•     }
• }
```

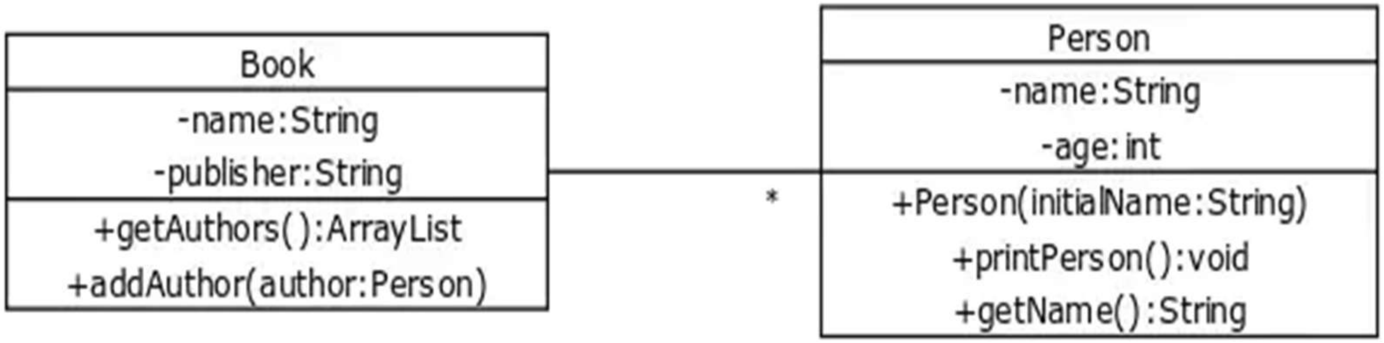
Connections between classes

- We could add the type of the elements in the ArrayList ArrayList<Person> and a label describing the connection "authors" to the class diagram above.


Book
-name:String
-publisher:String
+getAuthors():ArrayList
+addAuthor(author:Person)
*
Person
-name:String
-age:int
+Person(initialName:String)
+printPerson():void
+getName():String

Connections between classes

- If there is no arrowhead in a connection, both classes know about each other.
- Below is an example where a book knows about its author and a person knows about a book they have written.


Book
-name:String
-publisher:String
+getAuthors():ArrayList
+addAuthor(author:Person)
*
Person
-name:String
-age:int
+Person(initialName:String)
+printPerson():void
+getName():String

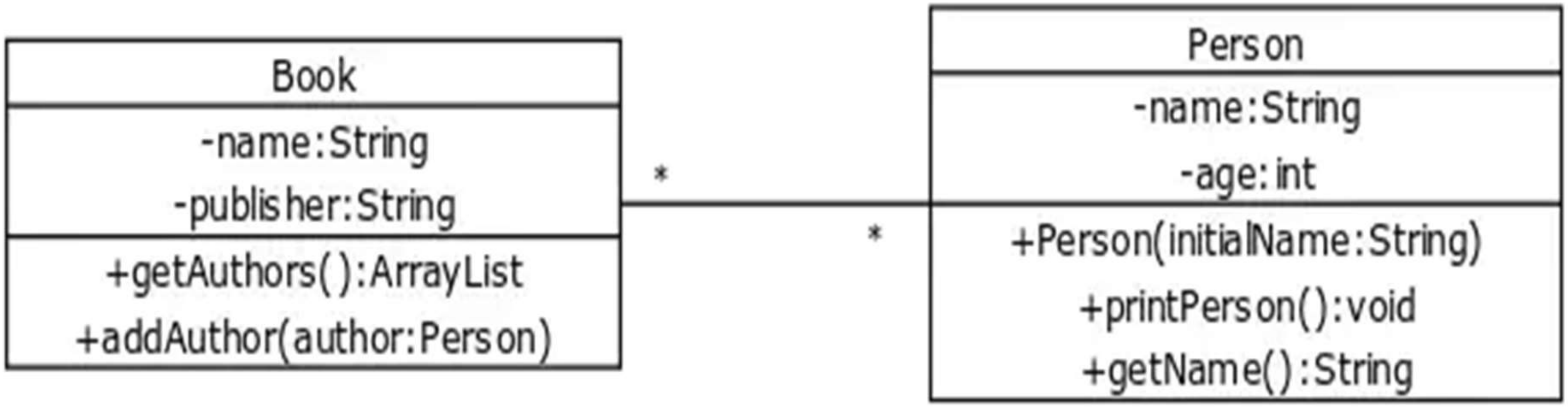
Connections between classes

```
• public class Person {
•     private String name;
•     private int age;
•     private Book book;

•     // ...
• }
```

```
• public class Book {
•     private String name;
•     private String publisher;
•     private ArrayList<Person> authors;

•     // ..
• }
```

Connections between classes

- As you can see, by default — if there is no star on the connection — the connection is singular.
- The classes above are interesting, because a Person can only have one book.
- If a person can have multiple books and a book can have multiple authors, we add a star to both ends of the connection:


Book
-name:String
-publisher:String
+getAuthors():ArrayList
+addAuthor(author:Person)
*
*
Person
-name:String
-age:int
+Person(initialName:String)
+printPerson():void
+getName():String

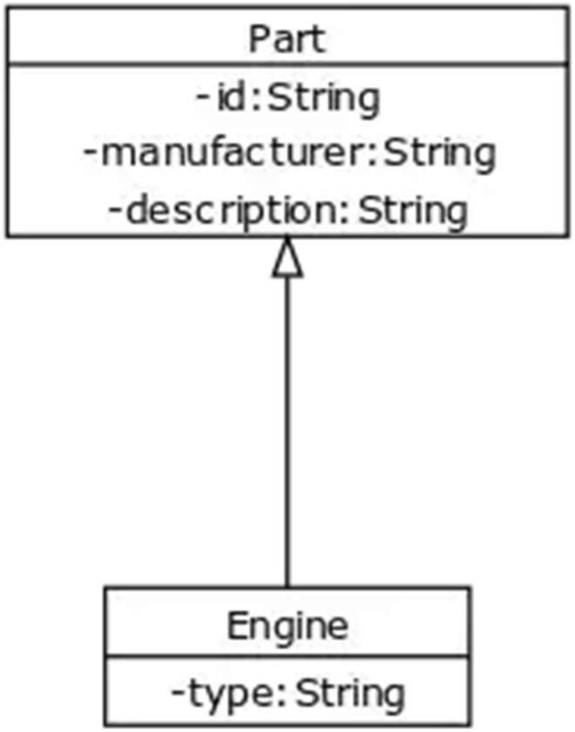
Connections between classes

- Now the person class would be as follows:

```
import java.util.ArrayList;

public class Person {
    private String name;
    private int age;
    private ArrayList<Book> books;

    // ...
}
```

Describing inheritance

- In a class diagram inheritance is described by an arrow with a triangle head.
- The triangle points to the class being inherited from.
- In the below example the Engine inherits the class Part.


Part
-id:String
-manufacturer:String
-description: String
Engine
-type: String

Describing inheritance

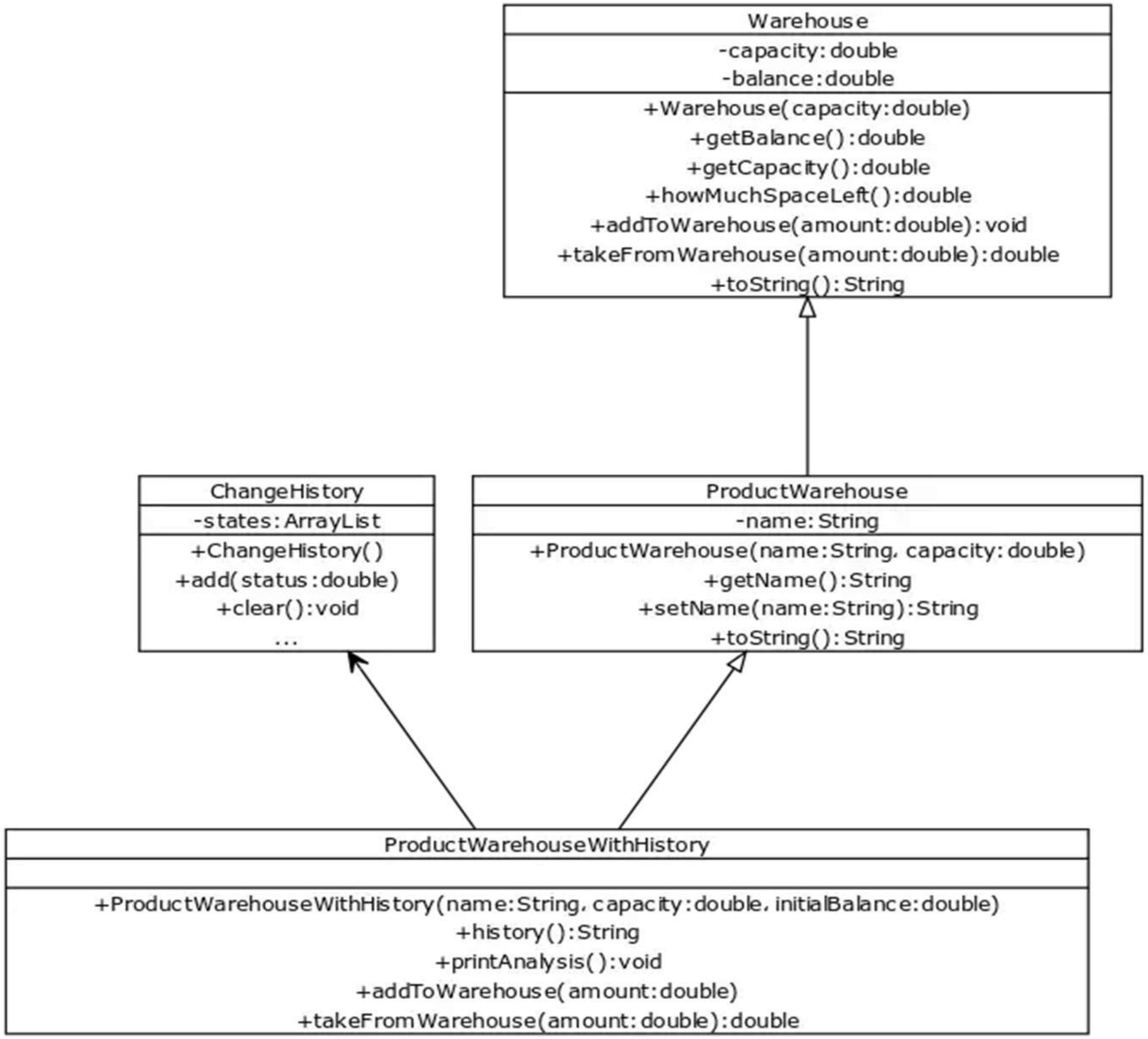
- In the below example the class diagram describes the classes from the Product warehouse.

- The ProductWarehouseWithHistory class inherits the ProductWarehouse class, which, in turn, inherits the Warehouse class.

- ChangeHistory is a separate class connected to the ProductWarehouse.

- ProductWarehouseWithHistory knows about the ChangeHistory but the ChangeHistory does not know about the ProductWarehouseWithHistory.

Describing inheritance



Warehouse

-capacity:double
-balance:double

+Warehouse(capacity:double)
+getBalance():double
+getCapacity():double
+howMuchSpaceLeft():double
+addToWarehouse(amount:double):void
+takeFromWarehouse(amount:double):double
+toString():String

ChangeHistory

-states:ArrayList

+ChangeHistory()
+add(status:double)
+clear():void
...

ProductWarehouse

-name:String

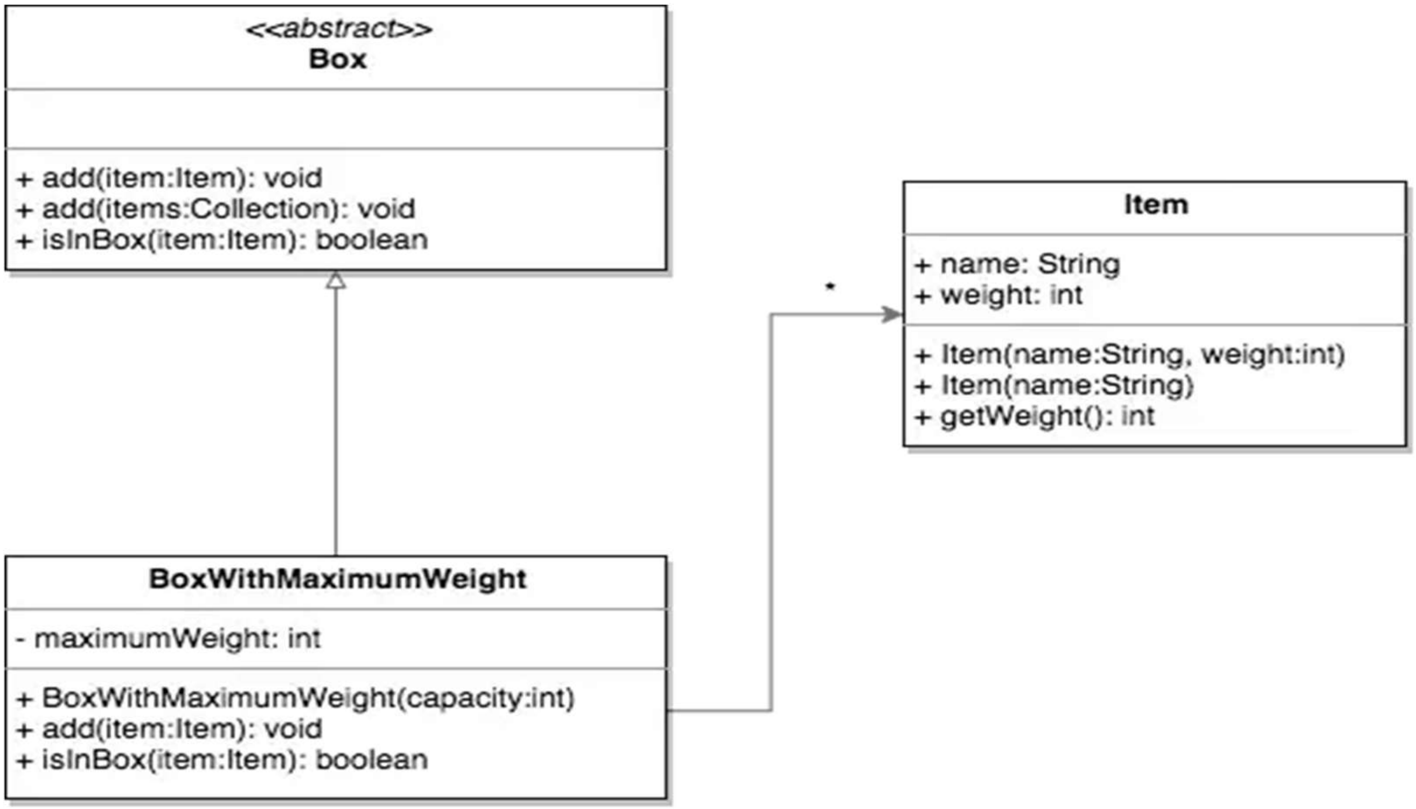
+ProductWarehouse(name:String, capacity:double)
+getName():String
+setName(name:String):String
+toString():String

ProductWarehouseWithHistory

+ProductWarehouseWithHistory(name:String, capacity:double, initialBalance:double)
+history():String
+printAnalysis():void
+addToWarehouse(amount:double)
+takeFromWarehouse(amount:double):double

Describing inheritance

- Inheritance of abstract classes is described almost the same way as regular classes.
- However we add the description <<abstract>> above the name of the class.
- The name of the class and its abstract methods are also written in cursive.



<<abstract>>
Box

+ add(item:Item): void
+ add(items:Collection): void
+ isInBox(item:Item): boolean

Item

+ name: String
+ weight: int

+ Item(name:String, weight:int)
+ Item(name:String)
+ getWeight(): int

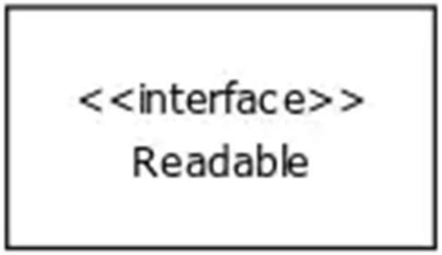
BoxWithMaximumWeight

- maximumWeight: int

+ BoxWithMaximumWeight(capacity:int)
+ add(item:Item): void
+ isInBox(item:Item): boolean

Describing interfaces

- In class diagrams, interfaces are written <<interface>> NameOfTheInterface.
- Below we describe an interface Readable.
- public interface Readable {
- }

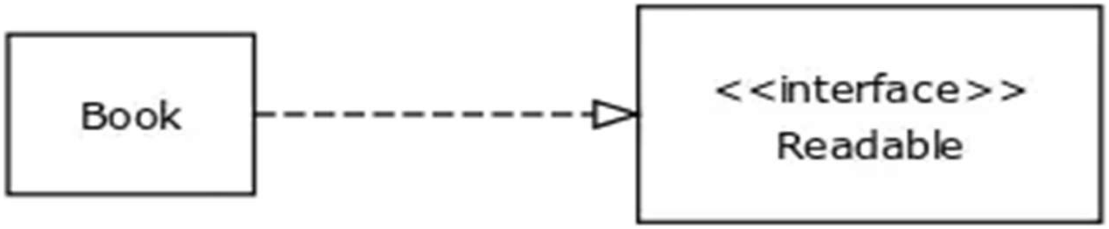


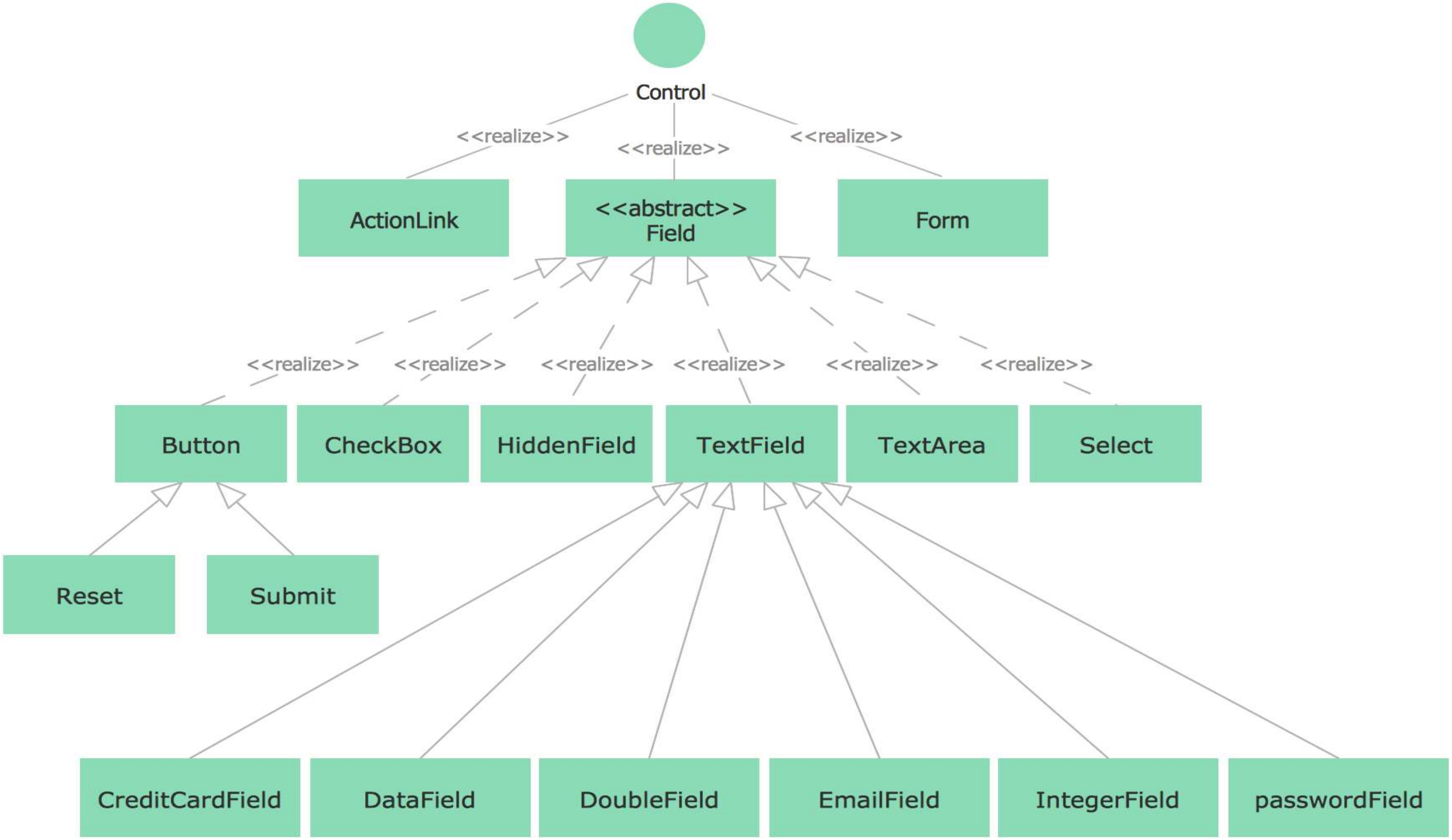
Describing interfaces

- Methods are described just like they are for a class.

- Implementing an interface is shown as a dashed arrow with a triangle arrowhead.

- Below, we describe a situation where Book implements interface Readable.

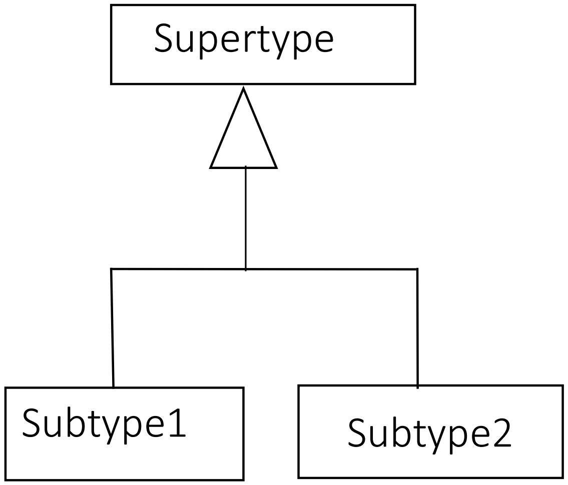



Control
<<realize>>
<<realize>>
<<realize>>
ActionLink
<<abstract>> Field
Form
<<realize>>
<<realize>>
<<realize>>
<<realize>>
<<realize>>
<<realize>>
Button
CheckBox
HiddenField
TextField
TextArea
Select
Reset
Submit
CreditCardField
DataField
DoubleField
EmailField
IntegerField
passwordField

OO Relationships

- There are two kinds of Relationships
  - Generalization (parent-child relationship)
  - Association (student enrolls in course)
- Associations can be further classified as
  - Aggregation
  - Composition

OO Relationships: Generalization

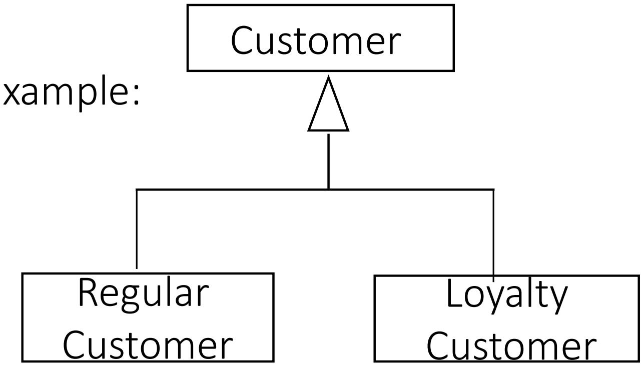


Supertype

Subtype1

Subtype2

Example:



Customer

Regular Customer

Loyalty Customer

-Inheritance is a required feature of object orientation

-Generalization expresses a parent/child relationship among related classes.

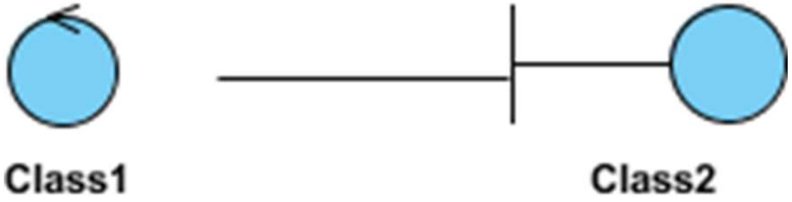
-Used for abstracting details in several layers

OO Relationships: Association

- Represent relationship between instances of classes
  - Student enrolls in a course
  - Courses have students
  - Courses have exams
  - Etc.
- Association has two ends
  - Role names (e.g. enrolls)
  - Multiplicity (e.g. One course can have many students)
  - Navigability (unidirectional, bidirectional)

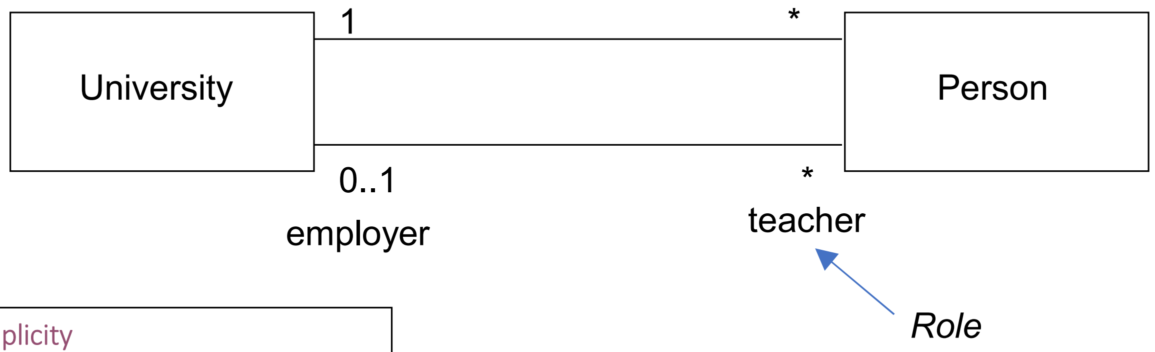
OO Relationships: Simple Association

- A structural link between two peer classes.



- There is an association between Class1 and Class2
- The figure below shows an example of simple association.
- There is an association that connects the <<control>> class Class1 and <<boundary>> class Class2.
- The relationship is displayed as a solid line connecting the two classes.

Association: Multiplicity and Roles



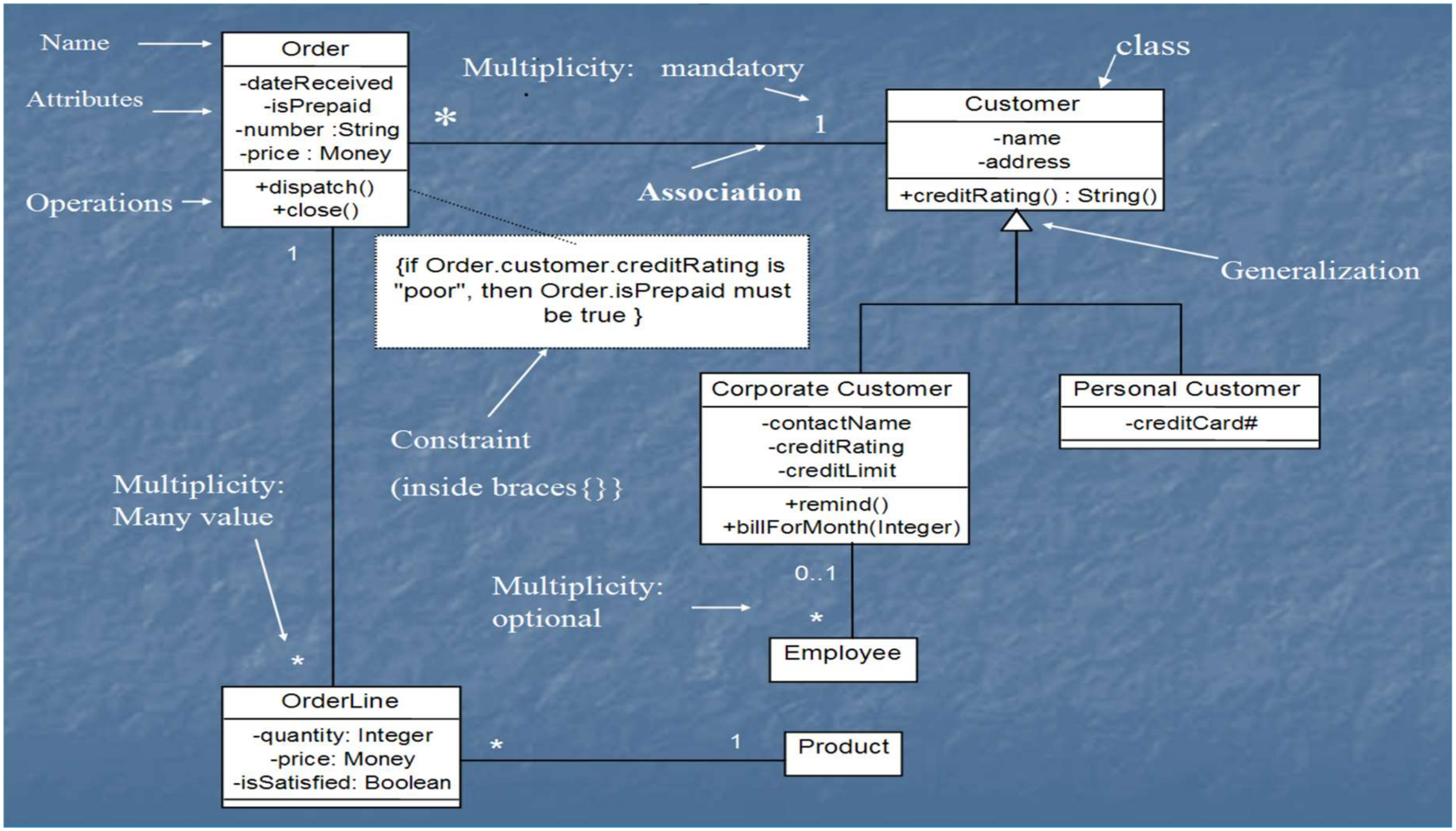
Multiplicity

Symbol	Meaning
1	One and only one
0..1	Zero or one
M..N	From M to N (natural language)
*	From zero to any positive integer
0..*	From zero to any positive integer
1..*	From one to any positive integer

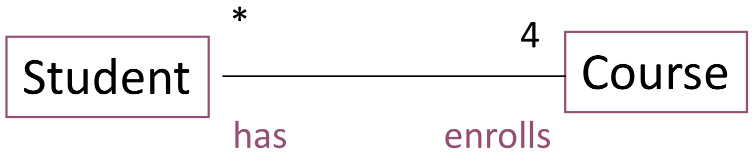
Role

"A given university groups many people; some act as students, others as teachers. A given student belongs to a single university; a given teacher may or may not be working for the university at a particular time."

Class diagram


Name
Attributes
Operations
Order
-dateReceived
-isPrepaid
-number :String
-price : Money
+dispatch()
+close()
Multiplicity: mandatory
*
1
Association
class
Customer
-name
-address
+creditRating() : String()
Generalization
{if Order.customer.creditRating is "poor", then Order.isPrepaid must be true }
Constraint
(inside braces{}}
1
Multiplicity: Many value
*
Corporate Customer
-contactName
-creditRating
-creditLimit
+remind()
+billForMonth(Integer)
Personal Customer
-creditCard#
Multiplicity: optional
0..1
*
Employee
OrderLine
-quantity: Integer
-price: Money
-isSatisfied: Boolean
*
1
Product

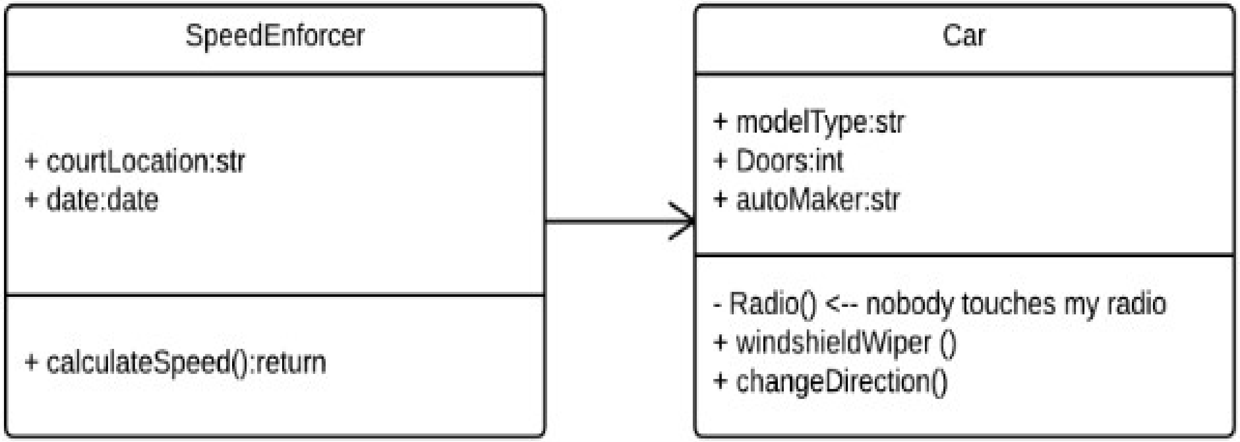
Association: Model to Implementation


Student
*
4
Course
has
enrolls

```
Class Student {
   Course enrolls[4];
}

Class Course {
   Student have[];
}
```

Association:

- **Unidirectional association:** A slightly less common relationship between two classes.
- One class is aware of the other and interacts with it.
- Unidirectional association is modeled with a straight connecting line that points an open arrowhead from the knowing class to the known class.


SpeedEnforcer
+ courtLocation:str
+ date:date
+ calculateSpeed():return
Car
+ modelType:str
+ Doors:int
+ autoMaker:str
- Radio() <-- nobody touches my radio
+ windshieldWiper ()
+ changeDirection()

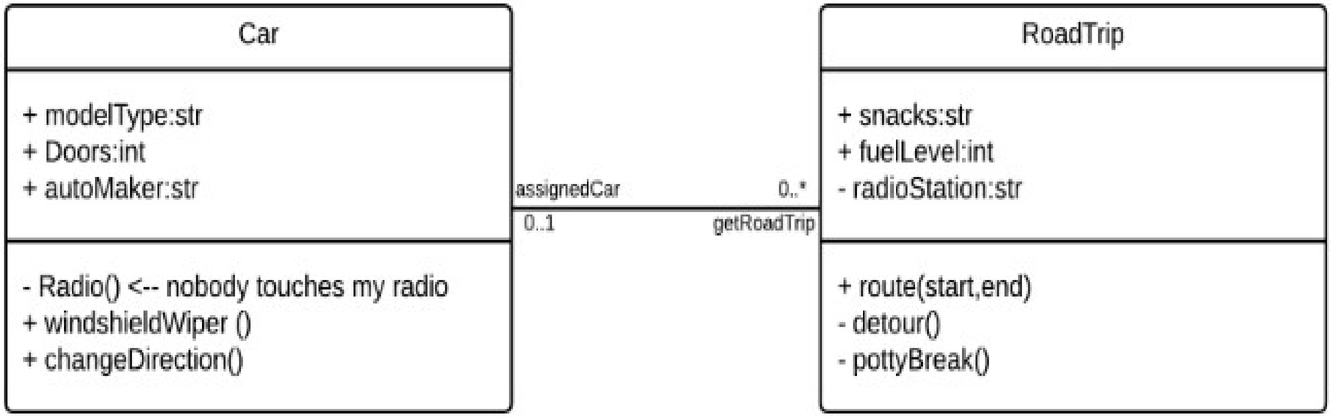
Association:

- As an example, on your road trip through New Delhi, you might run across a speed trap where a speed cam records your driving activity, but you won't know about it until you get a notification in the mail.

- It isn't drawn in the image, but in this case, the multiplicity value would be 0..* depending on how many times you drive by the speed cam.

Association:

- Bidirectional association: The default relationship between two classes.
- Both classes are aware of each other and their relationship with the other.
- This association is represented by a straight line between two classes.


Car
+ modelType:str
+ Doors:int
+ autoMaker:str
- Radio() <-- nobody touches my radio
+ windshieldWiper ()
+ changeDirection()
assignedCar
0..1
0..*
getRoadTrip
RoadTrip
+ snacks:str
+ fuelLevel:int
- radioStation:str
+ route(start,end)
- detour()
- pottyBreak()

Association:

- In the example, the Car class and RoadTrip class are interrelated.

- At one end of the line, the Car takes on the association of "assignedCar" with the multiplicity value of 0..1, so when the instance of RoadTrip exists, it can either have one instance of Car associated with it or no Cars associated with it.

- In this case, a separate Caravan class with a multiplicity value of 0..* is needed to demonstrate that a RoadTrip could have multiple instances of Cars associated with it.

- Since one Car instance could have multiple "getRoadTrip" associations—in other words, one car could go on multiple road trips—the multiplicity value is set to 0..*

OO Relationships: Composition

Association

Models the part–whole relationship

Composition

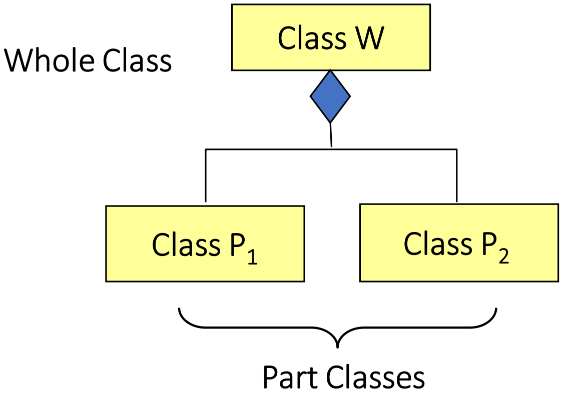
Also models the part–whole relationship but, in addition, Every part may belong to only one whole, and If the whole is deleted, so are the parts

Example:

A number of different chess boards:

Each square belongs to only one board.

If a chess board is thrown away, all 64 squares on that board go as well.



Whole Class

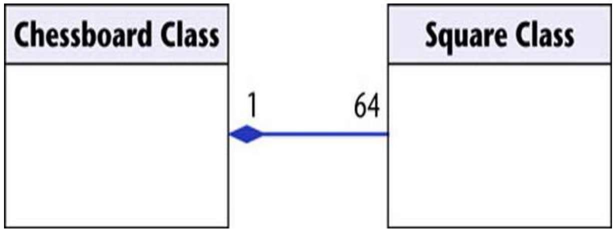
Class W

Class P_1

Class P_2

Part Classes

Example



Chessboard Class

Square Class

OO Relationships: Aggregation

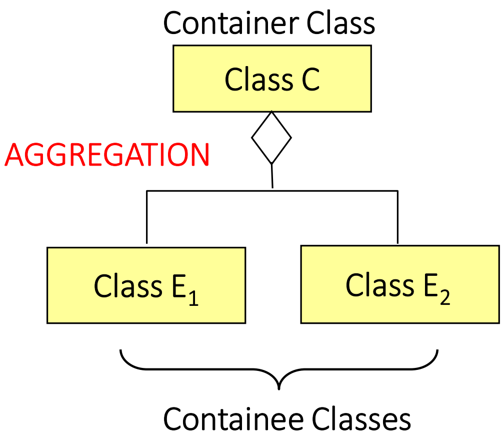
Aggregation:

expresses a relationship among instances of related classes.

It is a specific kind of Container-Containee relationship.

express a more informal relationship than composition expresses.

Aggregation is appropriate when Container and Containees have no special access privileges to each other.



Container Class

Class C

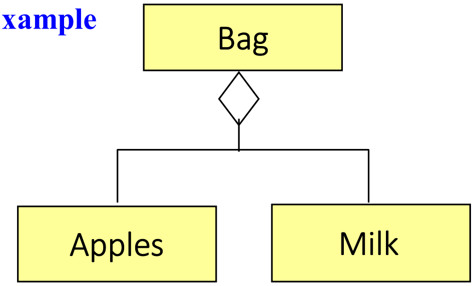
AGGREGATION

Class E_1

Class E_2

Containee Classes

Example

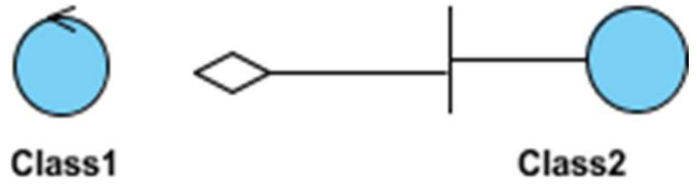


Bag

Apples

Milk

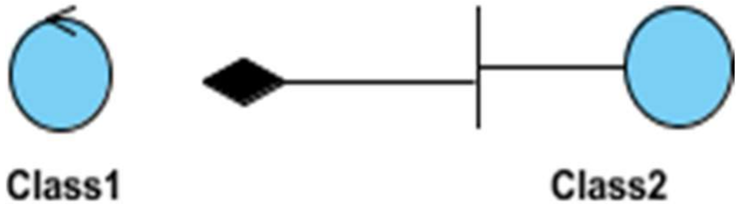
OO Relationships: Aggregation



- A special type of association.
- It represents a "part of" relationship.
- Class2 is part of Class1.
- Many instances (denoted by the *) of Class2 can be associated with Class1.
- Objects of Class1 and Class2 have separate lifetimes.
- The figure below shows an example of aggregation.
- The relationship is displayed as a solid line with a unfilled diamond at the association end, which is connected to the class that represents the aggregate.

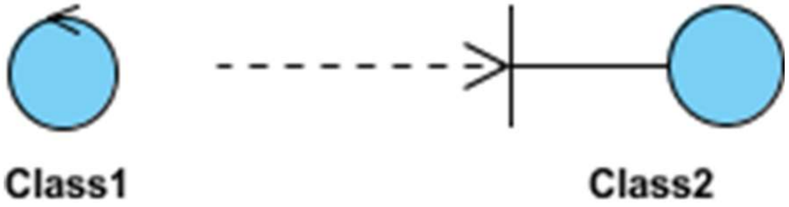
OO Relationships: Composition

- A special type of aggregation where parts are destroyed when the whole is destroyed.
- Objects of Class2 live and die with Class1.
- Class2 cannot stand by itself.
- The figure below shows an example of composition.
- The relationship is displayed as a solid line with a filled diamond at the association end, which is connected to the class that represents the whole or composite.



OO Relationships: Dependency

- An object of one class might use an object of another class in the code of a method.
- If the object is not stored in any field, then this is modeled as a dependency relationship.
- A special type of association.
- Exists between two classes if changes to the definition of one may cause changes to the other (but not the other way around).
- Class1 depends on Class2

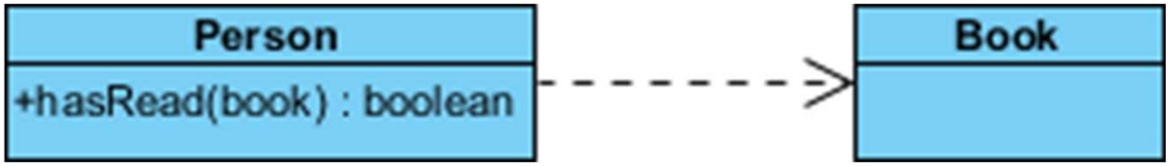


- The figure shows an example of dependency. The relationship is displayed as a dashed line with an open arrow.

OO Relationships: Dependency

- The figure below shows another example of dependency.

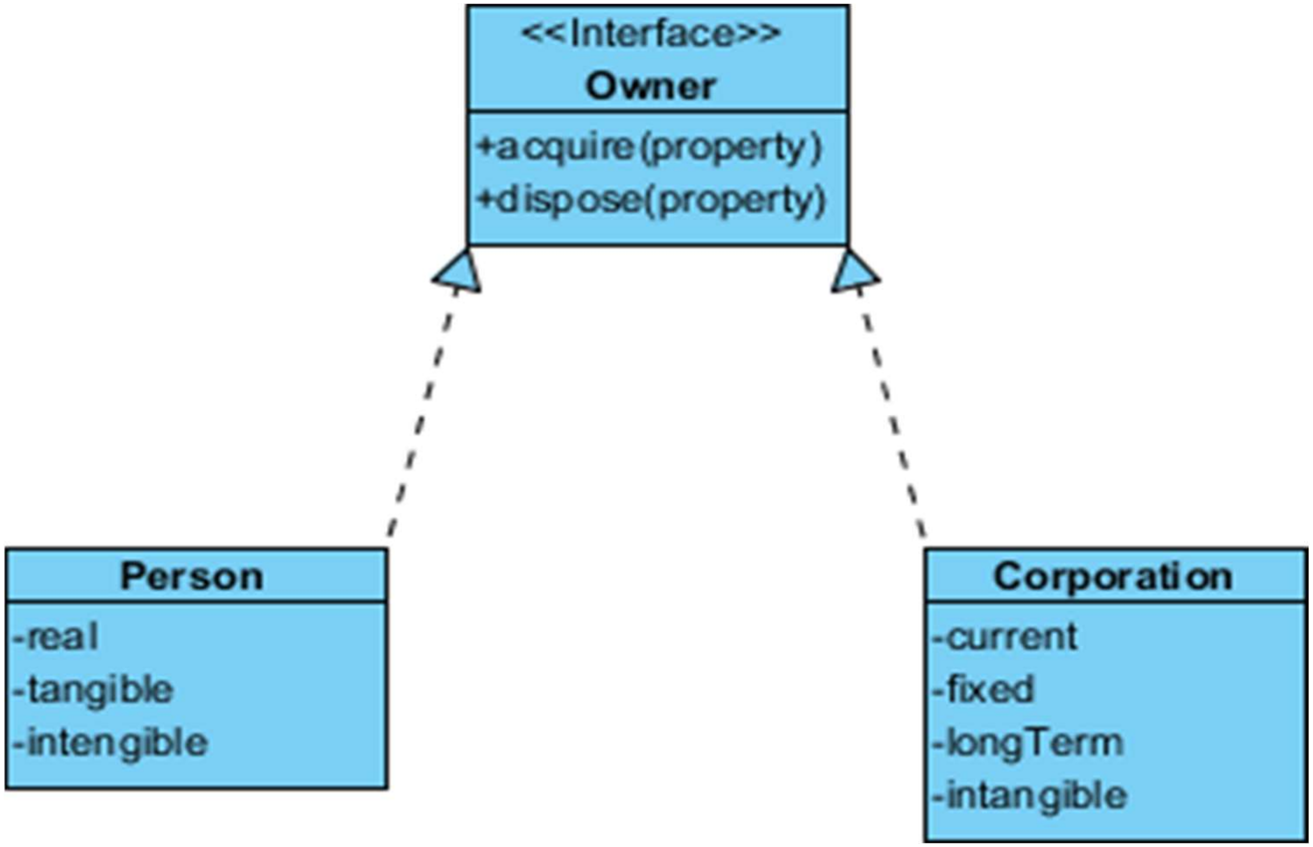
- The Person class might have a hasRead method with a Book parameter that returns true if the person has read the book (perhaps by checking some database)..


Person
+hasRead(book) : boolean
Book

OO Relationships: Realization

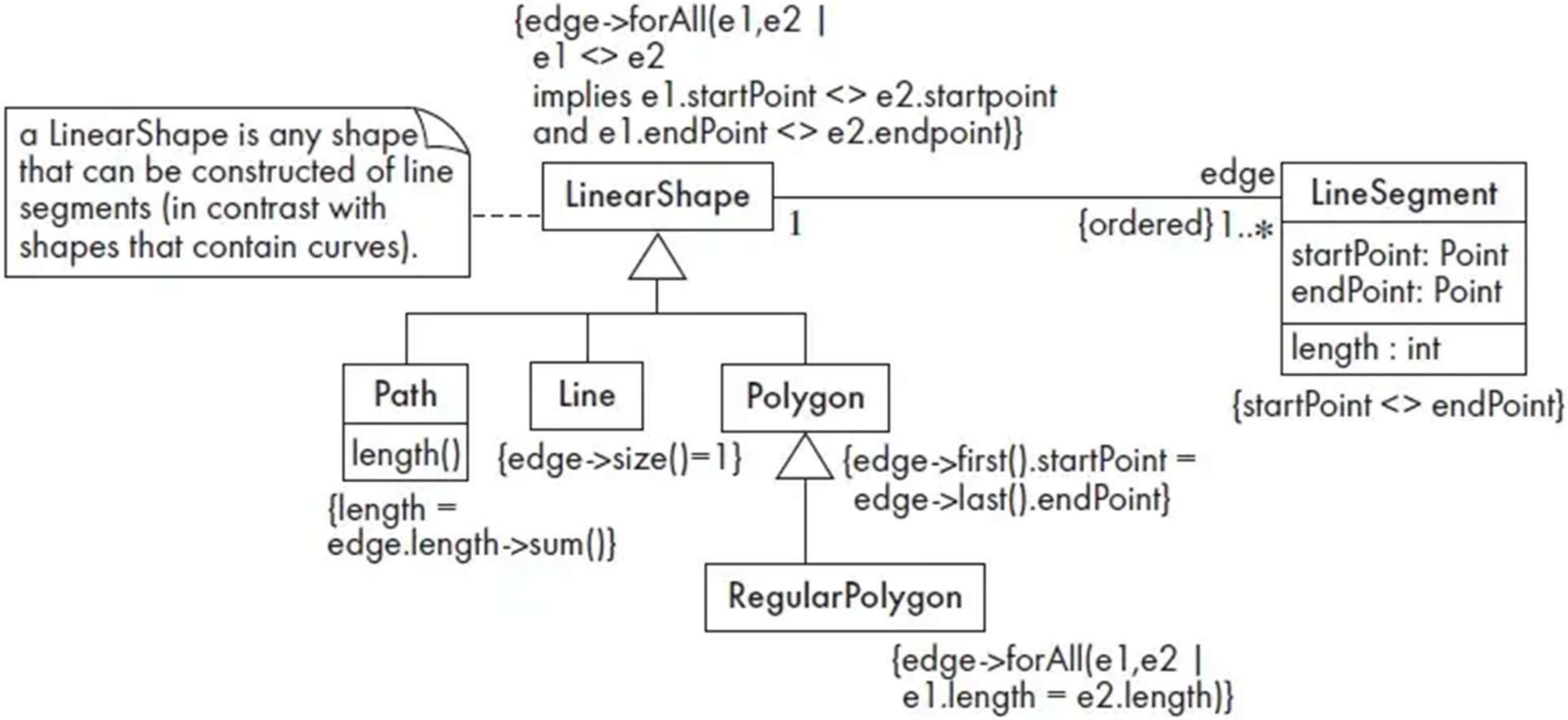
- Realization is a relationship between the blueprint class and the object containing its respective implementation level details.
- This object is said to realize the blueprint class.
- In other words, you can understand this as the relationship between the interface and the implementing class.
- For example, the Owner interface might specify methods for acquiring property and disposing of property.
- The Person and Corporation classes need to implement these methods, possibly in very different ways.

OO Relationships: Realization


<<Interface>>
Owner
+acquire(property)
+dispose(property)
Person
-real
-tangible
-intengible
Corporation
-current
-fixed
-longTerm
-intangible

Constraints

A formal structure is used to specify a constraint. They are shown in curly brackets. A constraint is a logical statement.


{edge->forAll(e1,e2 |
e1 <> e2
implies e1.startPoint <> e2.startpoint
and e1.endPoint <> e2.endpoint)}
a LinearShape is any shape that can be constructed of line segments (in contrast with shapes that contain curves).
LinearShape
edge
1
{ordered} 1..*
LineSegment
startPoint: Point
endPoint: Point
length : int
{startPoint <> endPoint}
Path
length()
{length =
edge.length->sum()}
Line
{edge->size()=1}
Polygon
{edge->first().startPoint =
edge->last().endPoint}
RegularPolygon
{edge->forAll(e1,e2 |
e1.length = e2.length)}

Aggregation vs. Composition

Composition is really a strong form of association

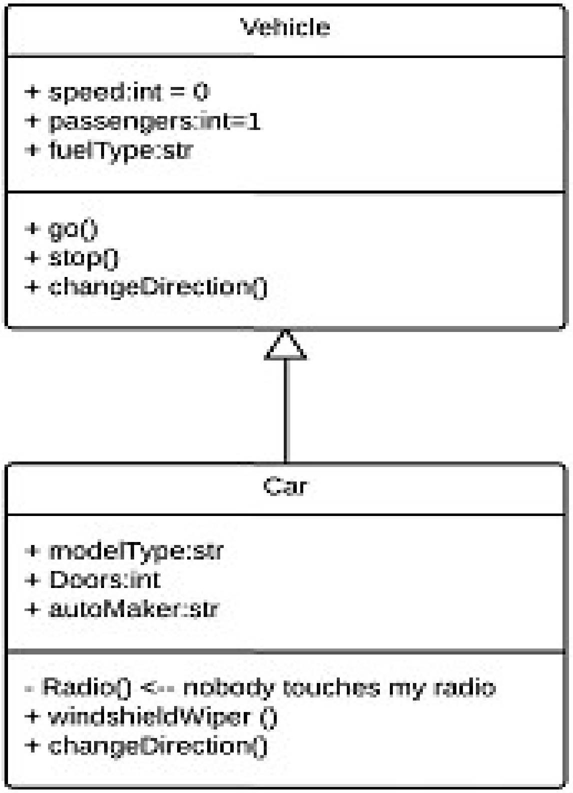
- components have only one owner
- components cannot exist independent of their owner
- components live or die with their owner
- e.g. Each car has an engine that can not be shared with other cars.

Aggregations

- may form "part of" the association, but may not be essential to it.
- They may also exist independent of the aggregate.
- e.g. Apples may exist independent of the bag.

Inheritance: The process of a child or sub-class taking on the functionality of a parent or superclass, also known as generalization.

It's symbolized with a straight connected line with a closed arrowhead pointing towards the superclass.

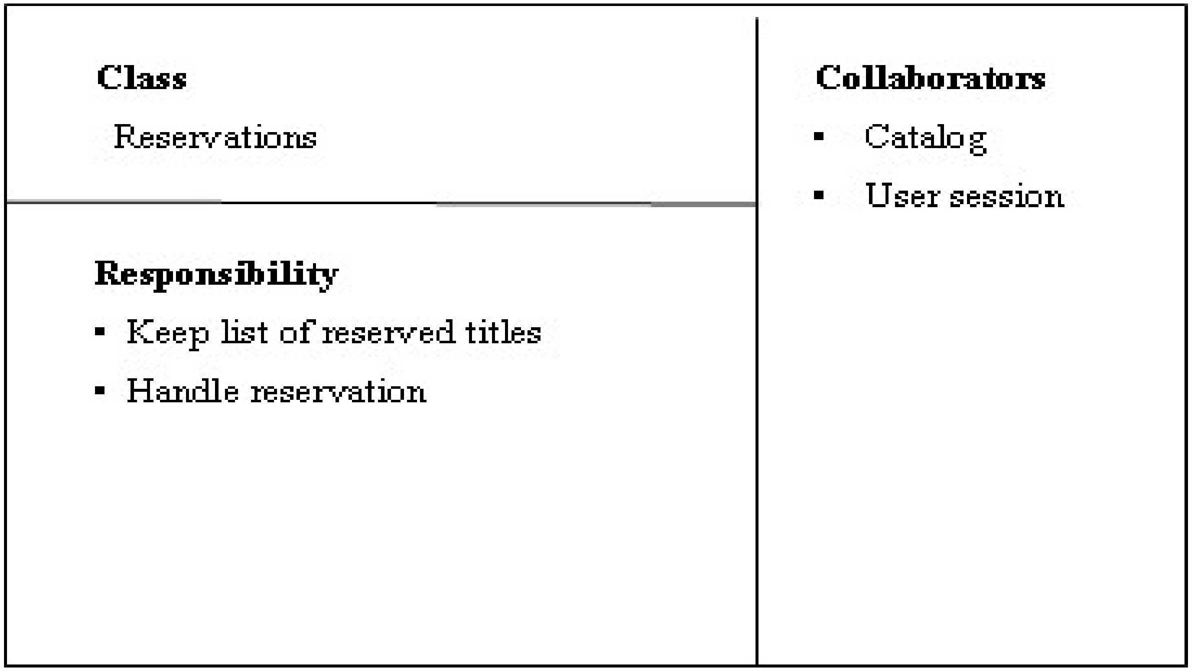

Vehicle
+ speed:int = 0
+ passengers:int=1
+ fuelType:str
+ go()
+ stop()
+ changeDirection()
Car
+ modelType:str
+ Doors:int
+ autoMaker:str
- Radio() <-- nobody touches my radio
+ windshieldWiper ()
+ changeDirection()

In this example, the object "Car" would inherit all of the attributes (speed, numbers of passengers, fuel) and methods (go(), stop(), changeDirection()) of the parent class ("Vehicle") in addition to the specific attributes (model type, number of doors, auto maker) and methods of its own class (Radio(), windshieldWiper(), ac/heat()). Inheritance is shown in a class diagram by using a solid line with a closed, hollow arrow.

Good Practice: CRC Card

Class Responsibility Collaborator

- easy to describe how classes work by moving cards around; allows to quickly consider alternatives.



Class

Reservations

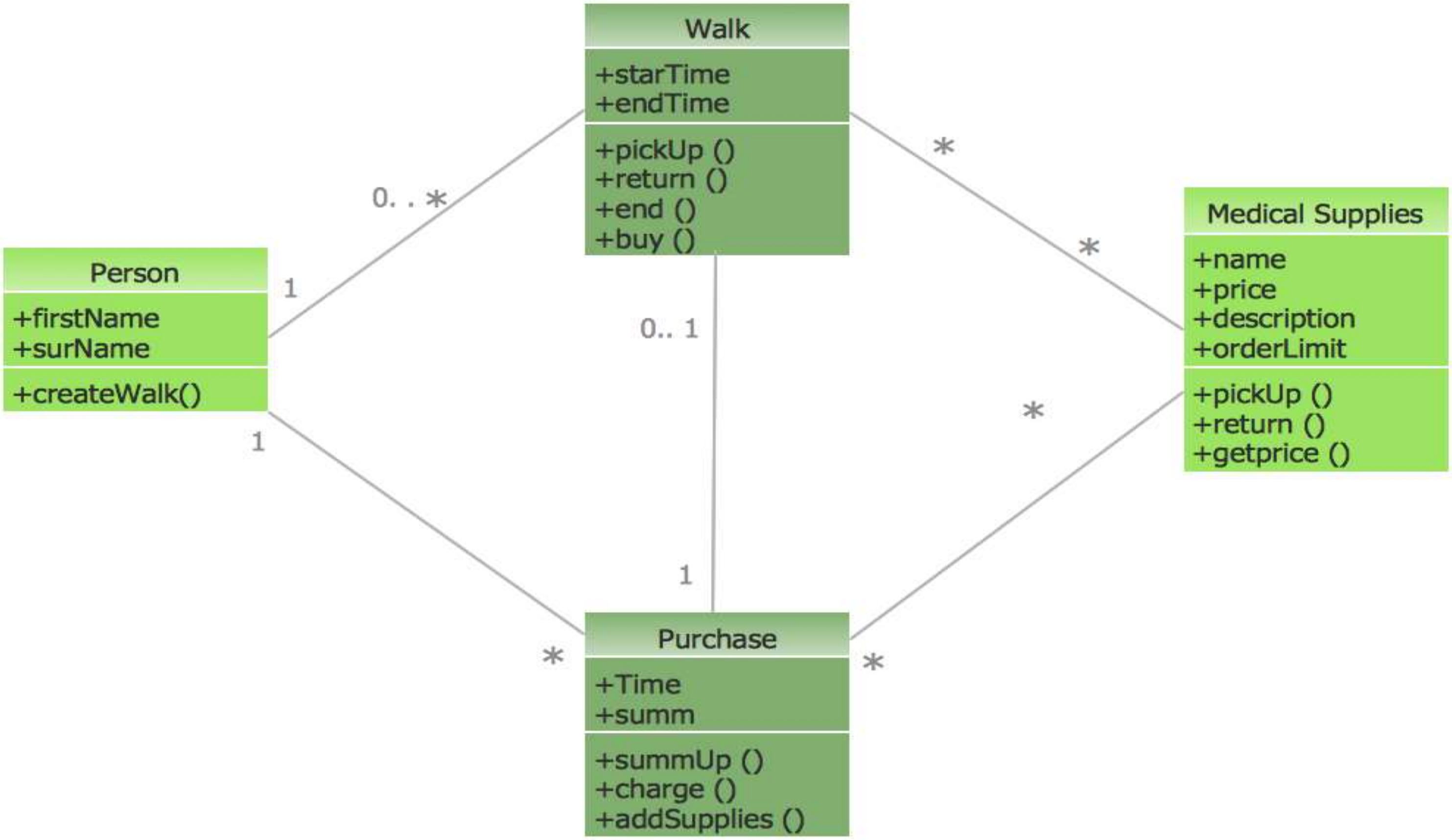
Responsibility

- Keep list of reserved titles
- Handle reservation

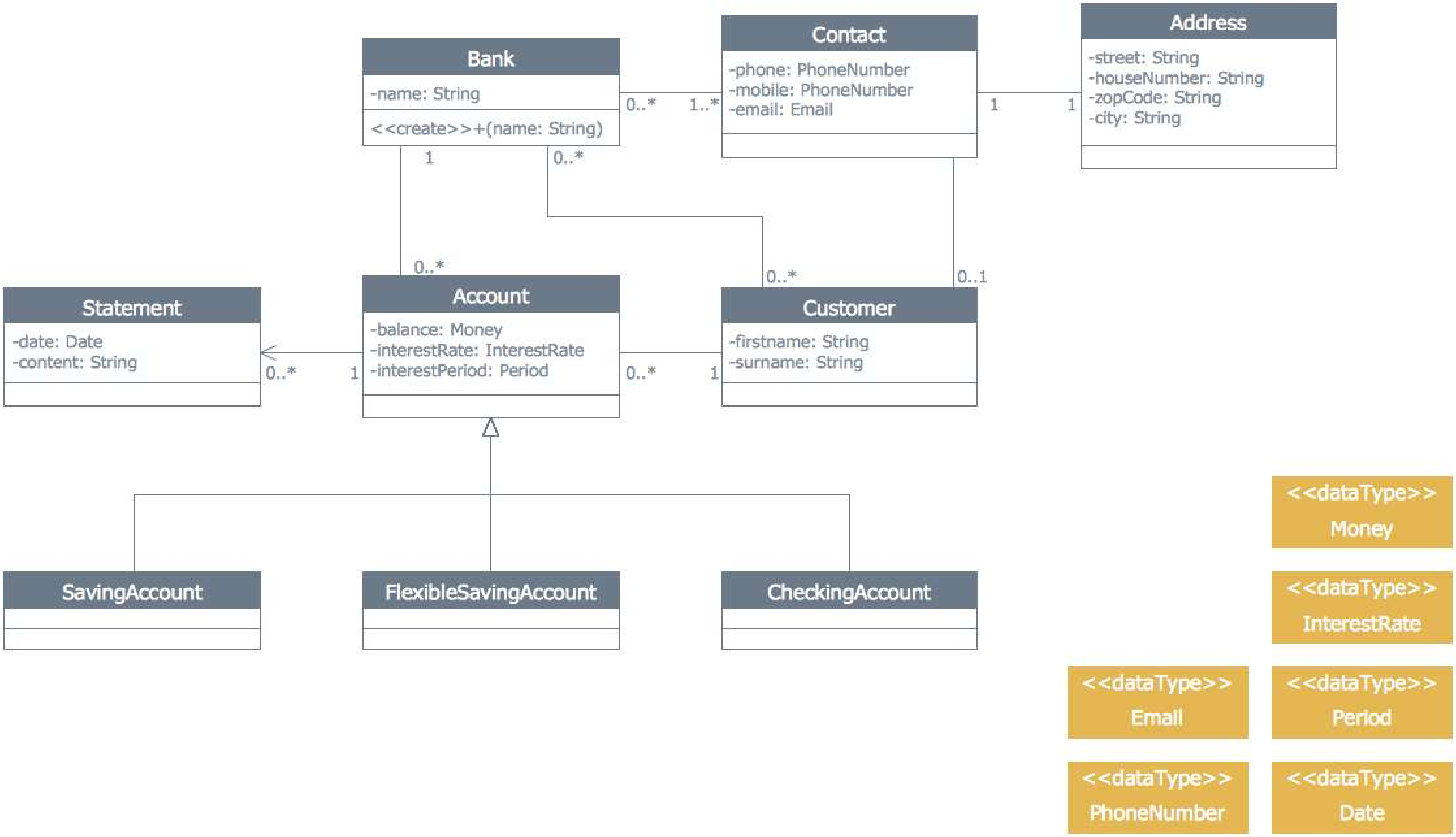
Collaborators

- Catalog
- User session

- This sample shows the work of the medical shop and is used at the creation of the medical shop or pharmacy project, for understanding the process of shopping in the medical shop, for keeping purchases, for medical supplies.


Walk
+starTime
+endTime
+pickUp ()
+return ()
+end ()
+buy ()
Person
+firstName
+surName
+createWalk()
Medical Supplies
+name
+price
+description
+orderLimit
+pickUp ()
+return ()
+getprice ()
Purchase
+Time
+summ
+summUp ()
+charge ()
+addSupplies ()
0. . *
1
*
*
0.. 1
1
*
1
*
*

Banking system and ATMs



Bank
-name: String
<<create>>+(name: String)

Contact
-phone: PhoneNumber
-mobile: PhoneNumber
-email: Email

Address
-street: String
-houseNumber: String
-zopCode: String
-city: String

Statement
-date: Date
-content: String

Account
-balance: Money
-interestRate: InterestRate
-interestPeriod: Period

Customer
-firstname: String
-surname: String

SavingAccount

FlexibleSavingAccount

CheckingAccount

<<dataType>>
Money

<<dataType>>
InterestRate

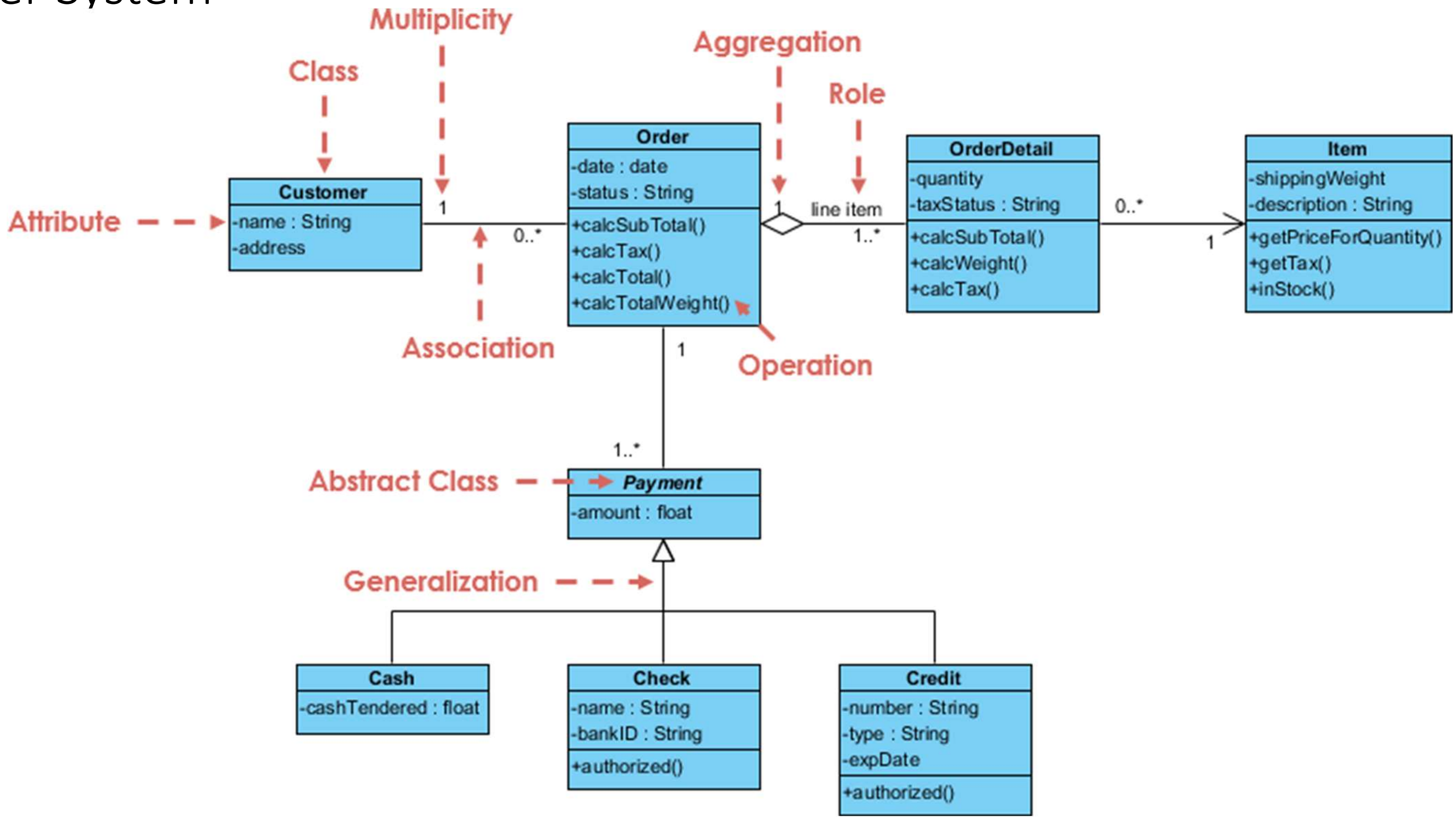
<<dataType>>
Email

<<dataType>>
Period

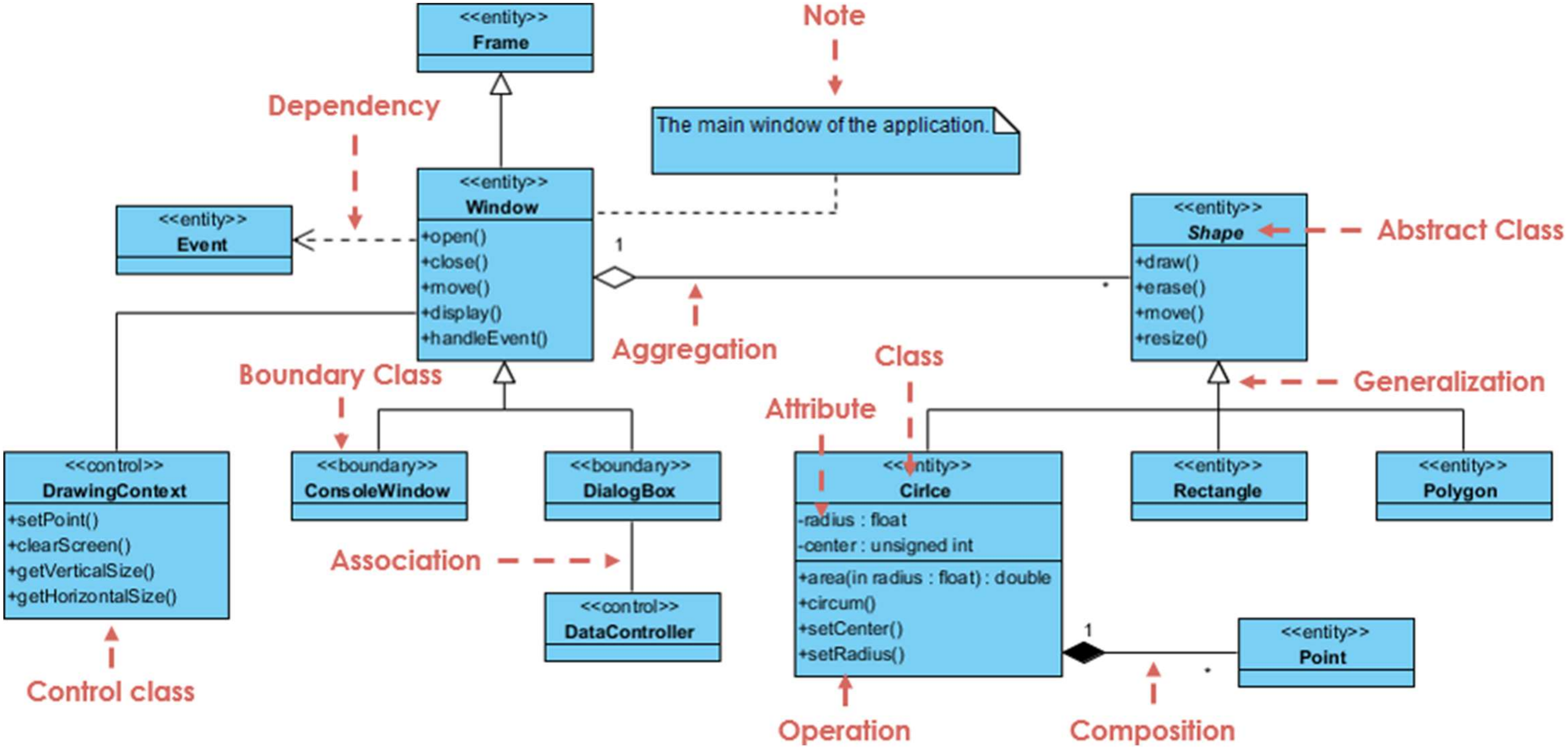
<<dataType>>
PhoneNumber

<<dataType>>
Date

Order System


Multiplicity
Class
Aggregation
Role
Attribute
Association
Operation
Abstract Class
Generalization
Customer
-name : String
-address
Order
-date : date
-status : String
+calcSubTotal()
+calcTax()
+calcTotal()
+calcTotalWeight()
OrderDetail
-quantity
-taxStatus : String
+calcSubTotal()
+calcWeight()
+calcTax()
Item
-shippingWeight
-description : String
+getPriceForQuantity()
+getTax()
+inStock()
Payment
-amount : float
Cash
-cashTendered : float
Check
-name : String
-bankID : String
+authorized()
Credit
-number : String
-type : String
-expDate
+authorized()
1
0..*
1
line item
1..*
0..*
1
1
1..*

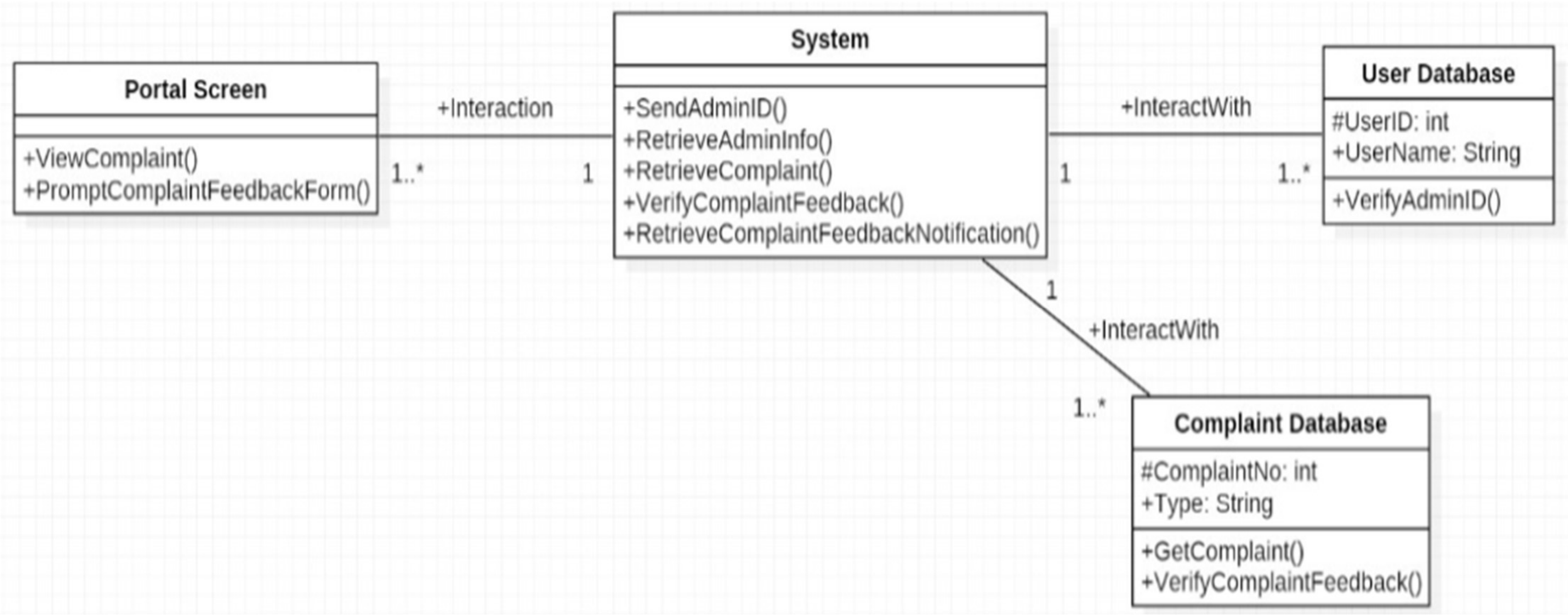
GUI - A class diagram may also have notes attached to classes or relationships.


Note
The main window of the application.
Dependency
<<entity>>
Frame
<<entity>>
Window
+open()
+close()
+move()
+display()
+handleEvent()
<<entity>>
Event
1
<<entity>>
Shape
+draw()
+erase()
+move()
+resize()
Abstract Class
Aggregation
Boundary Class
Class
Attribute
Generalization
<<control>>
DrawingContext
+setPoint()
+clearScreen()
+getVerticalSize()
+getHorizontalSize()
<<boundary>>
ConsoleWindow
<<boundary>>
DialogBox
<<entity>>
Cirlce
-radius : float
-center : unsigned int
+area(in radius : float) : double
+circum()
+setCenter()
+setRadius()
<<entity>>
Rectangle
<<entity>>
Polygon
Association
<<control>>
DataController
1
<<entity>>
Point
Control class
Operation
Composition

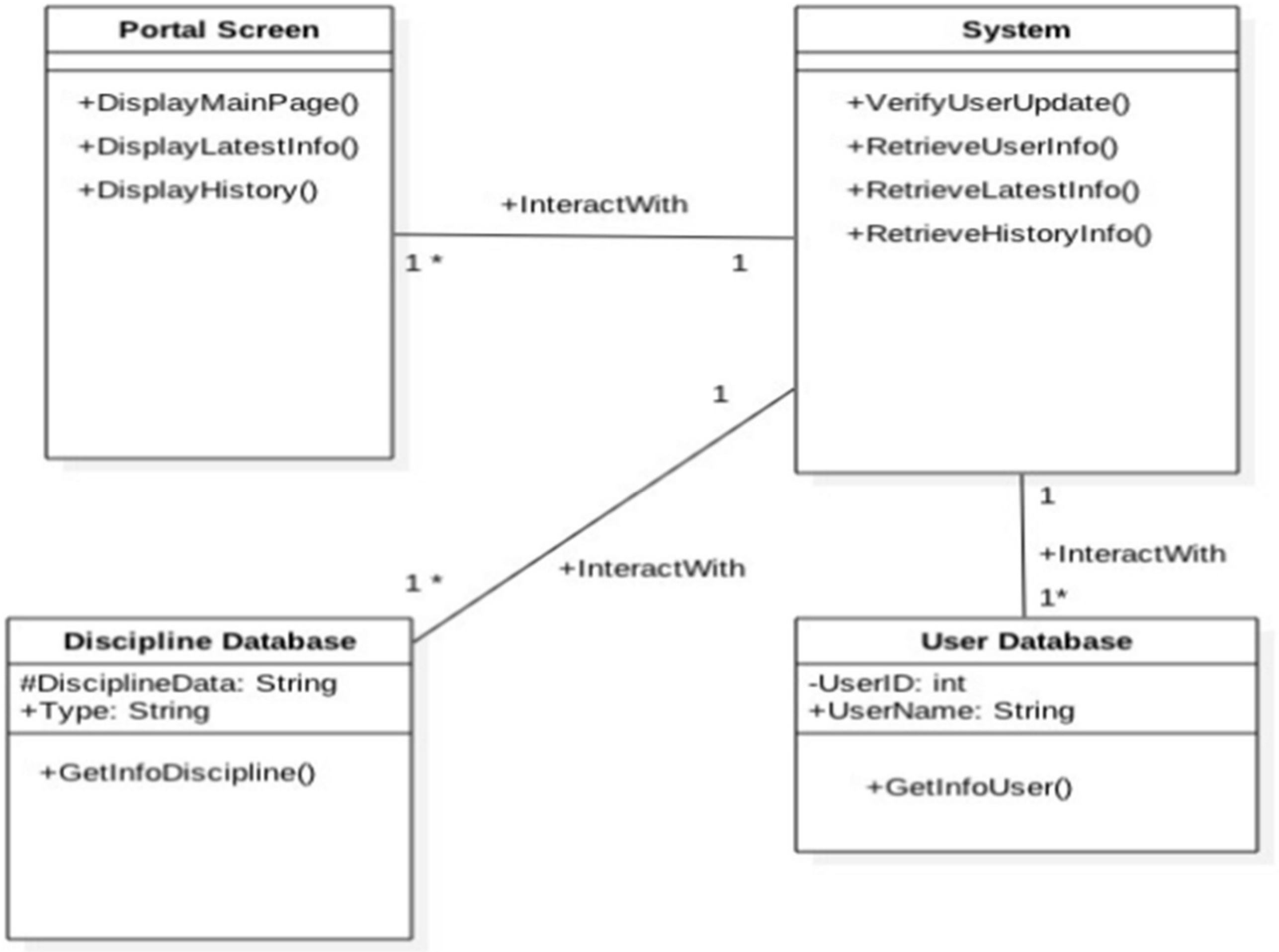
Students Facility System

- A university students facility system
- This portal was made with a goal to facilitate and to bring convenience to all the registered students
- Student can visit the portal
- They can make a request or complaint
- Manage complaint Section
- Manage disciplinary Section
- Manage system payment
- Manage data collection
- Manage suggestion Screen

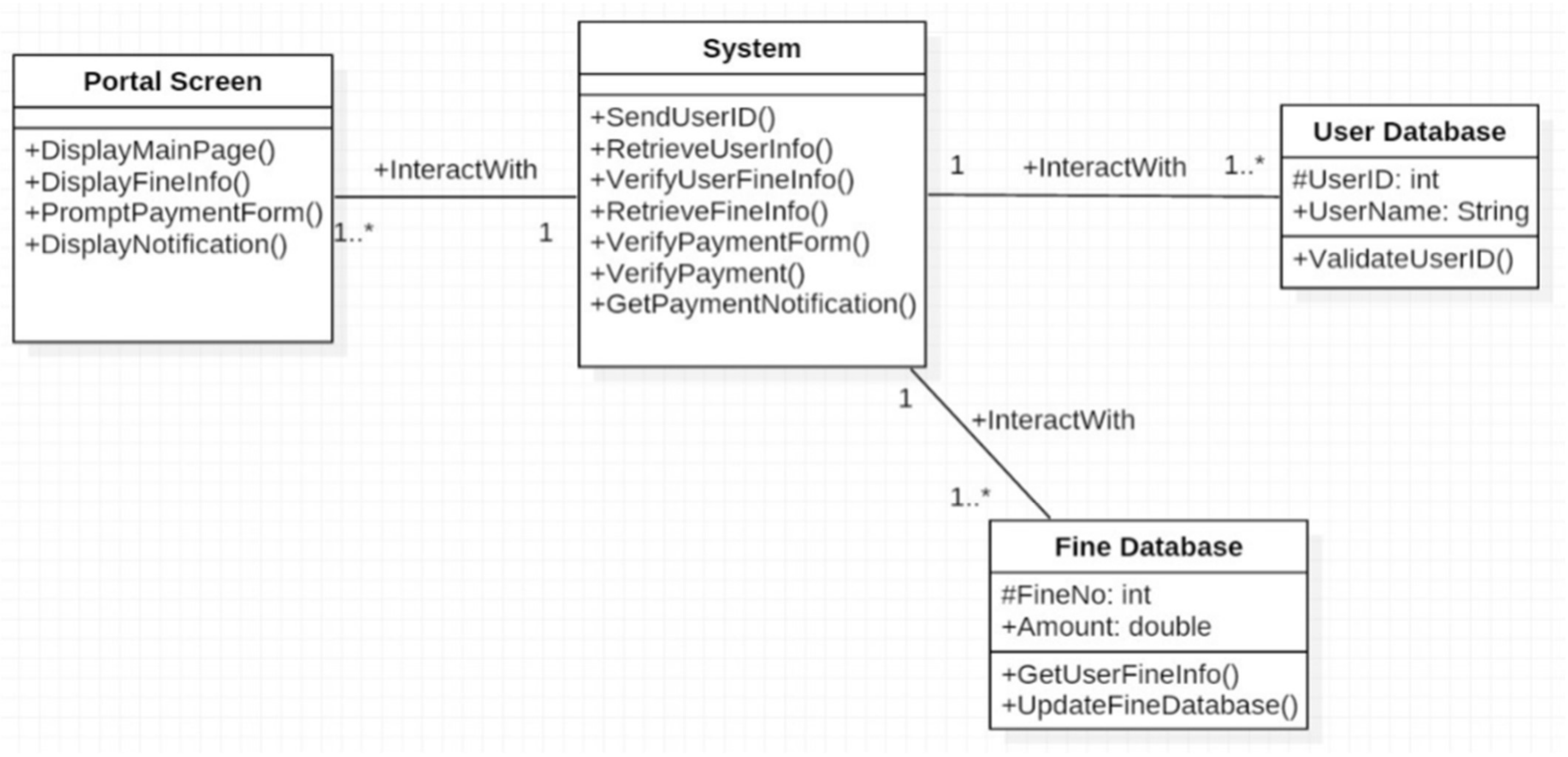
Manage complaint Section


Portal Screen
+ViewComplaint()
+PromptComplaintFeedbackForm()
+Interaction
1..*
1
System
+SendAdminID()
+RetrieveAdminInfo()
+RetrieveComplaint()
+VerifyComplaintFeedback()
+RetrieveComplaintFeedbackNotification()
+InteractWith
1
1..*
User Database
#UserID: int
+UserName: String
+VerifyAdminID()
1
+InteractWith
1..*
Complaint Database
#ComplaintNo: int
+Type: String
+GetComplaint()
+VerifyComplaintFeedback()

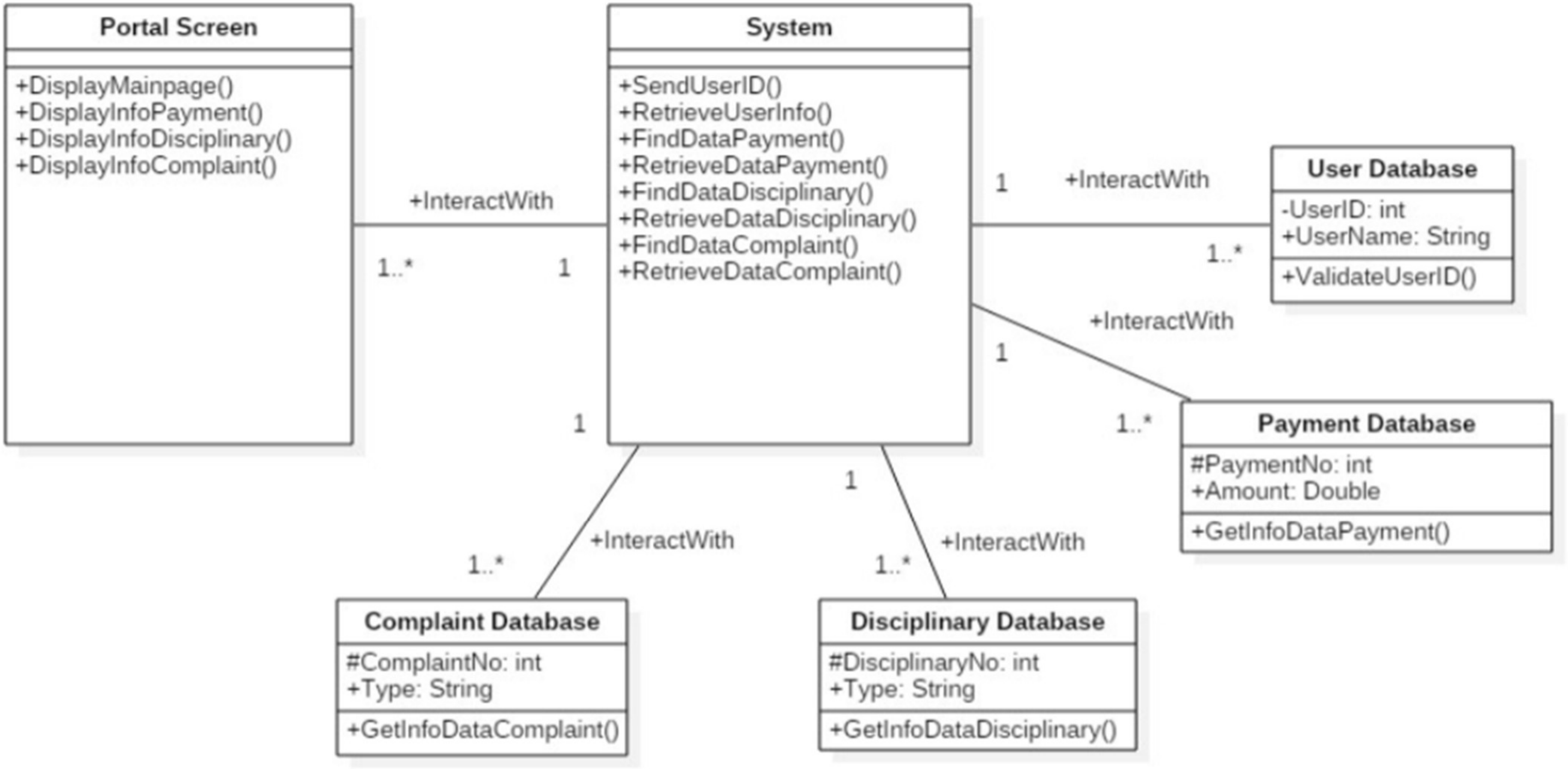
Manage Disciplinary Section


Portal Screen
+DisplayMainPage()
+DisplayLatestInfo()
+DisplayHistory()
System
+VerifyUserUpdate()
+RetrieveUserInfo()
+RetrieveLatestInfo()
+RetrieveHistoryInfo()
+InteractWith
1 *
1
1
+InteractWith
1 *
1
+InteractWith
1*
Discipline Database
#DisciplineData: String
+Type: String
+GetInfoDiscipline()
User Database
-UserID: int
+UserName: String
+GetInfoUser()

Manage System Payment


Portal Screen
+DisplayMainPage()
+DisplayFineInfo()
+PromptPaymentForm()
+DisplayNotification()
+InteractWith
1..*
1
System
+SendUserID()
+RetrieveUserInfo()
+VerifyUserFineInfo()
+RetrieveFineInfo()
+VerifyPaymentForm()
+VerifyPayment()
+GetPaymentNotification()
1
+InteractWith
1..*
User Database
#UserID: int
+UserName: String
+ValidateUserID()
1
+InteractWith
1..*
Fine Database
#FineNo: int
+Amount: double
+GetUserFineInfo()
+UpdateFineDatabase()

Manage Data Collection



Portal Screen

+DisplayMainpage()
+DisplayInfoPayment()
+DisplayInfoDisciplinary()
+DisplayInfoComplaint()

System

+SendUserID()
+RetrieveUserInfo()
+FindDataPayment()
+RetrieveDataPayment()
+FindDataDisciplinary()
+RetrieveDataDisciplinary()
+FindDataComplaint()
+RetrieveDataComplaint()

User Database

-UserID: int
+UserName: String

+ValidateUserID()

Payment Database

#PaymentNo: int
+Amount: Double

+GetInfoDataPayment()

Complaint Database

#ComplaintNo: int
+Type: String

+GetInfoDataComplaint()

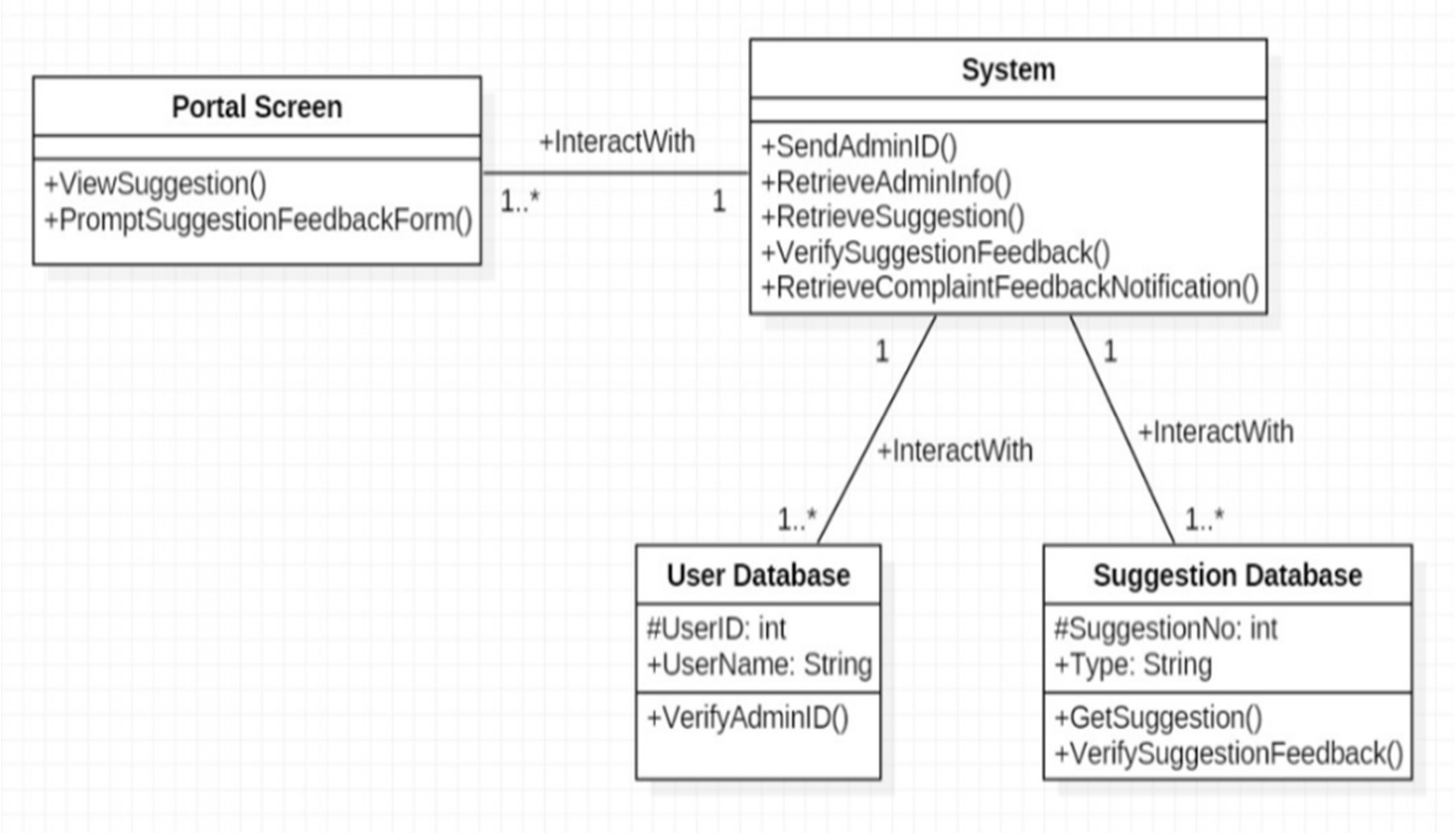
Disciplinary Database

#DisciplinaryNo: int
+Type: String

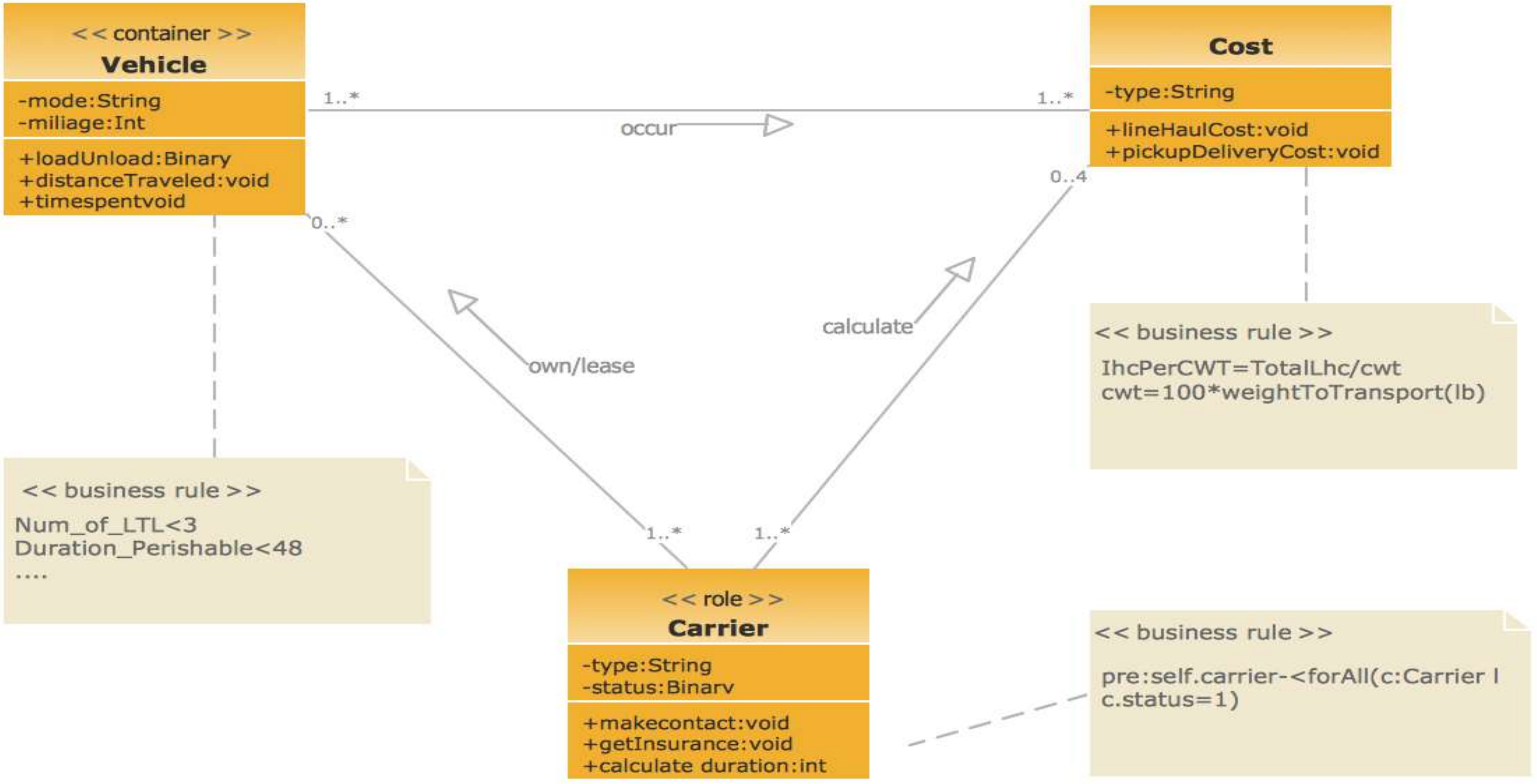
+GetInfoDataDisciplinary()

+InteractWith
1..*
1
1
+InteractWith
1..*
+InteractWith
1
1..*
1
+InteractWith
1..*
1
+InteractWith
1..*

Manage Suggestion Screen


Portal Screen
+ViewSuggestion()
+PromptSuggestionFeedbackForm()
+InteractWith
1..*
1
System
+SendAdminID()
+RetrieveAdminInfo()
+RetrieveSuggestion()
+VerifySuggestionFeedback()
+RetrieveComplaintFeedbackNotification()
1
1
+InteractWith
+InteractWith
1..*
1..*
User Database
#UserID: int
+UserName: String
+VerifyAdminID()
Suggestion Database
#SuggestionNo: int
+Type: String
+GetSuggestion()
+VerifySuggestionFeedback()

Class Diagram for Goods Transporter System



<< container >>
Vehicle

-mode:String
-miliage:Int

+loadUnload:Binary
+distanceTraveled:void
+timespentvoid

Cost

-type:String

+lineHaulCost:void
+pickupDeliveryCost:void

<< role >>
Carrier

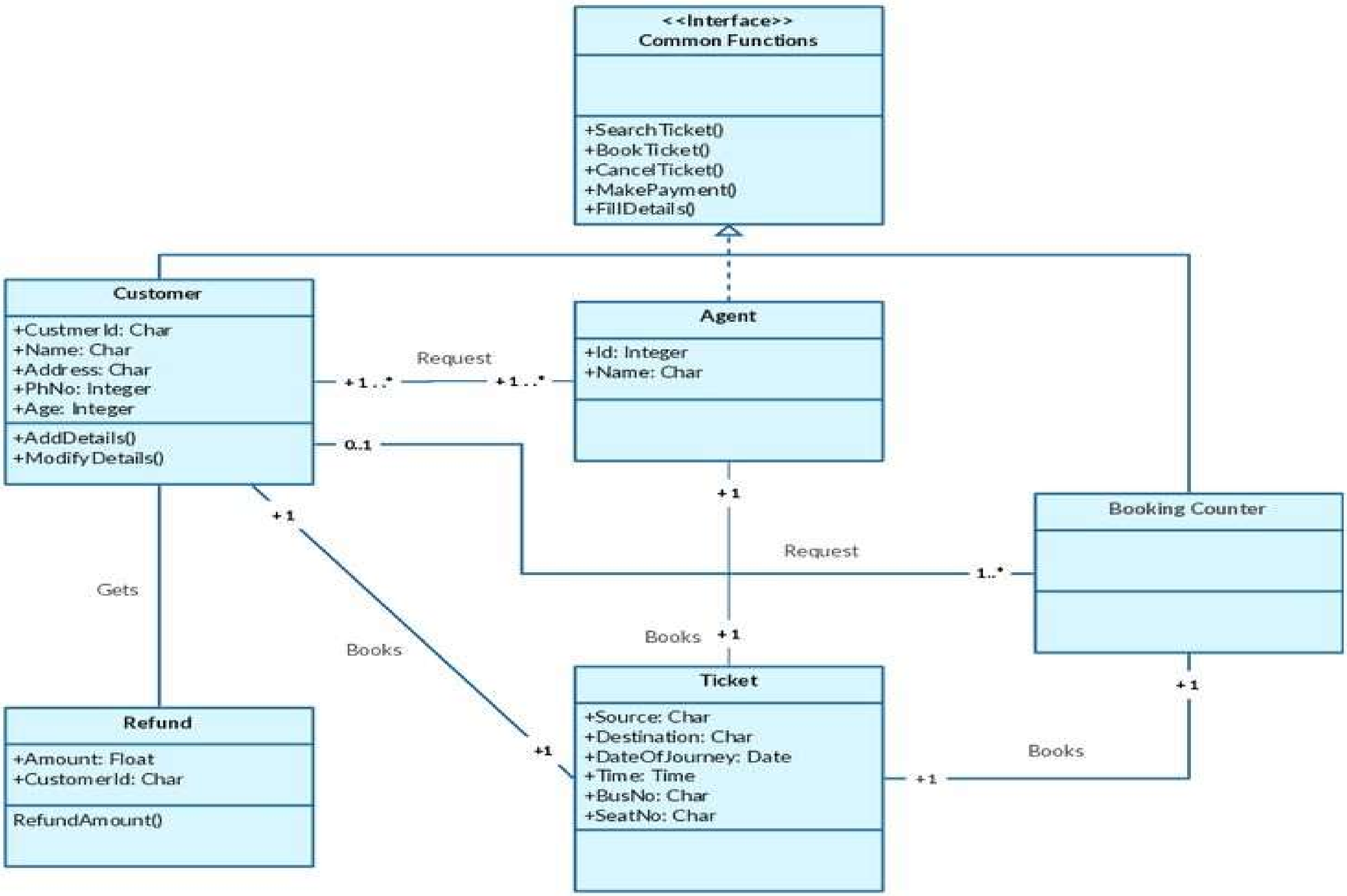
-type:String
-status:Binarv

+makecontact:void
+getInsurance:void
+calculate duration:int

<< business rule >>
Num_of_LTL<3
Duration_Perishable<48
....

<< business rule >>
IhcPerCWT=TotalLhc/cwt
cwt=100*weightToTransport(lb)

<< business rule >>
pre:self.carrier-<forAll(c:Carrier |
c.status=1)



<<Interface>>
Common Functions

+SearchTicket()
+BookTicket()
+CancelTicket()
+MakePayment()
+FillDetails()

Customer

+CustmerId: Char
+Name: Char
+Address: Char
+PhNo: Integer
+Age: Integer

+AddDetails()
+ModifyDetails()

Agent

+Id: Integer
+Name: Char

Booking Counter

Refund

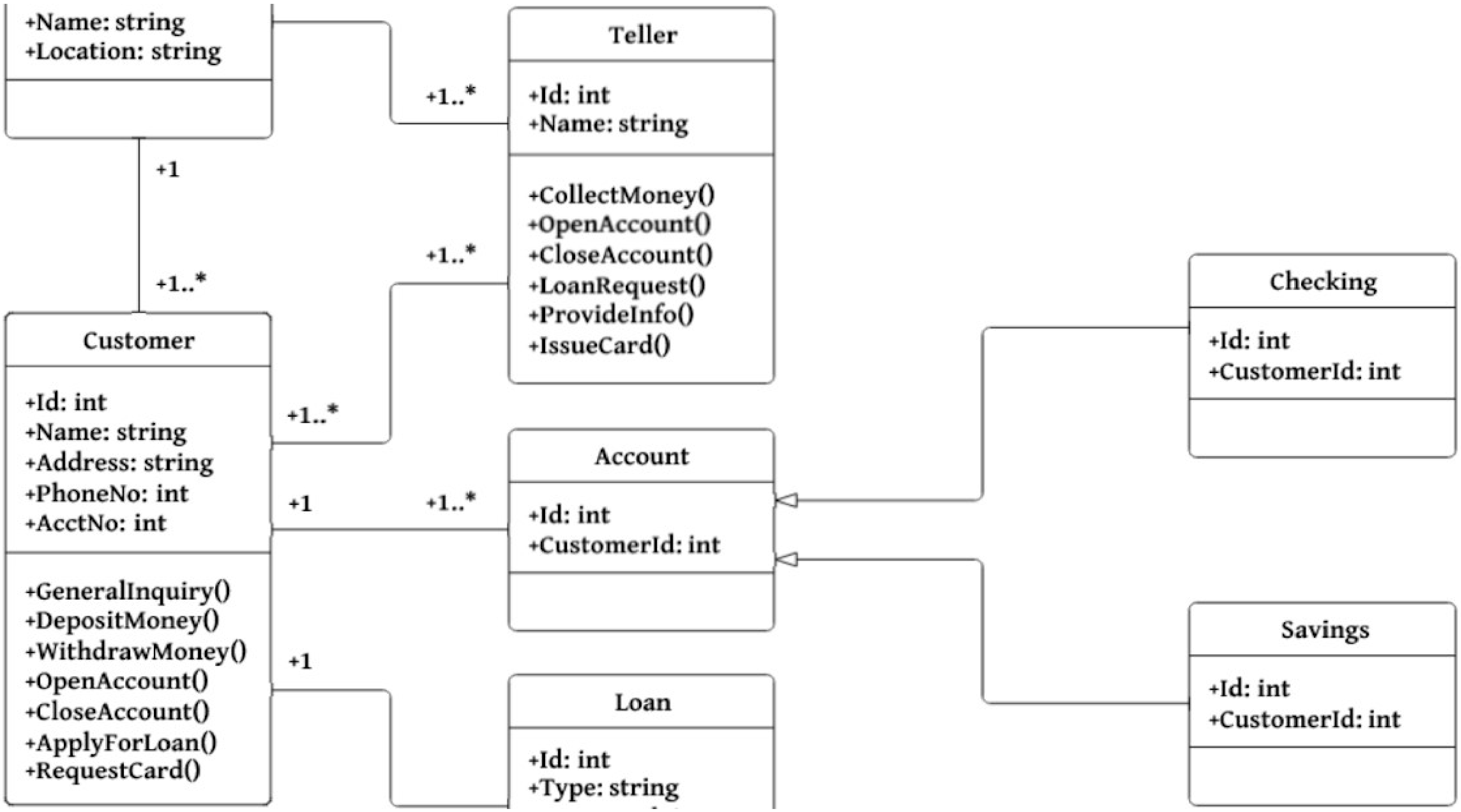
+Amount: Float
+CustomerId: Char

RefundAmount()

Ticket

+Source: Char
+Destination: Char
+DateOfJourney: Date
+Time: Time
+BusNo: Char
+SeatNo: Char

Request
+1..*
+1..*
0..1
Request
1..*
+1
+1
Gets
+1
Books
+1
Books
+1
Books
+1


+Name: string
+Location: string

Teller

+Id: int
+Name: string

+CollectMoney()
+OpenAccount()
+CloseAccount()
+LoanRequest()
+ProvideInfo()
+IssueCard()

Customer

+Id: int
+Name: string
+Address: string
+PhoneNo: int
+AcctNo: int

+GeneralInquiry()
+DepositMoney()
+WithdrawMoney()
+OpenAccount()
+CloseAccount()
+ApplyForLoan()
+RequestCard()

Account

+Id: int
+CustomerId: int

Loan

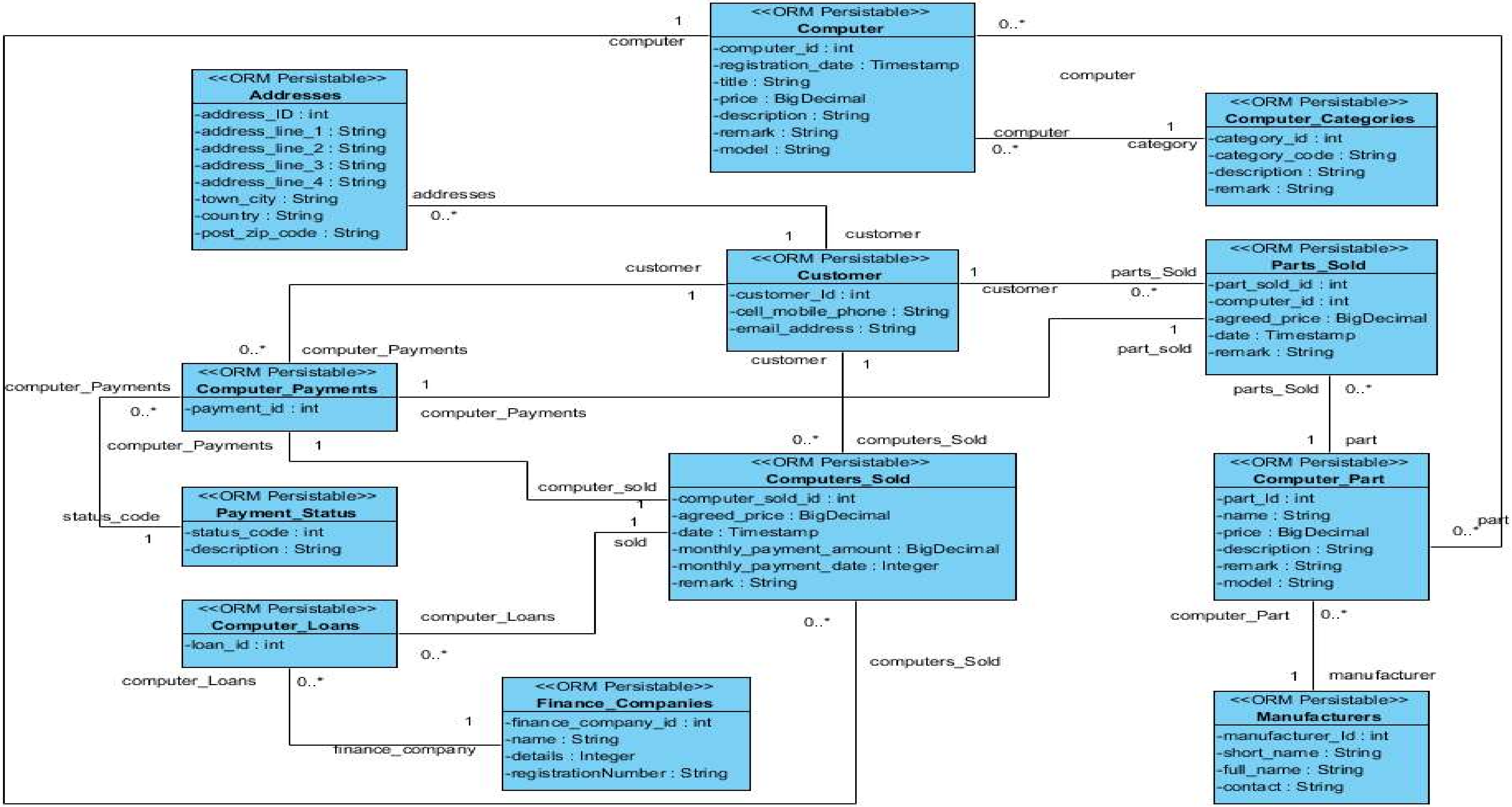
+Id: int
+Type: string

Checking

+Id: int
+CustomerId: int

Savings

+Id: int
+CustomerId: int



<<ORM Persistable>>
Computer
-computer_id : int
-registration_date : Timestamp
-title : String
-price : BigDecimal
-description : String
-remark : String
-model : String

<<ORM Persistable>>
Addresses
-address_ID : int
-address_line_1 : String
-address_line_2 : String
-address_line_3 : String
-address_line_4 : String
-town_city : String
-country : String
-post_zip_code : String

<<ORM Persistable>>
Computer_Categories
-category_id : int
-category_code : String
-description : String
-remark : String

<<ORM Persistable>>
Customer
-customer_Id : int
-cell_mobile_phone : String
-email_address : String

<<ORM Persistable>>
Parts_Sold
-part_sold_id : int
-computer_id : int
-agreed_price : BigDecimal
-date : Timestamp
-remark : String

<<ORM Persistable>>
Computer_Payments
-payment_id : int

<<ORM Persistable>>
Computers_Sold
-computer_sold_id : int
-agreed_price : BigDecimal
-date : Timestamp
-monthly_payment_amount : BigDecimal
-monthly_payment_date : Integer
-remark : String

<<ORM Persistable>>
Computer_Part
-part_Id : int
-name : String
-price : BigDecimal
-description : String
-remark : String
-model : String

<<ORM Persistable>>
Payment_Status
-status_code : int
-description : String

<<ORM Persistable>>
Computer_Loans
-loan_id : int

<<ORM Persistable>>
Finance_Companies
-finance_company_id : int
-name : String
-details : Integer
-registrationNumber : String

<<ORM Persistable>>
Manufacturers
-manufacturer_Id : int
-short_name : String
-full_name : String
-contact : String

1
computer
0..*
computer
computer
1
0..*
category
addresses
0..*
1
customer
customer
1
1
customer
parts_Sold
0..*
1
part_sold
0..*
computer_Payments
computer_Payments
0..*
1
computer_Payments
customer
1
parts_Sold
0..*
computer_Payments
1
0..*
computers_Sold
1
part
computer_sold
1
status_code
1
1
sold
0..*
part
computer_Loans
0..*
0..*
computers_Sold
computer_Part
0..*
computer_Loans
0..*
1
manufacturer
1
finance_company

Class diagrams - Summary

- Class diagrams are an excellent way to describe a problem and a problem-domain to others.

- They are particularly useful when a programmer is designing a program with multiple classes.

- Often a class diagram is drawn on a whiteboard or a large sheet of paper during the design phase.

- Class diagrams should be thought of as helpful tools to build a program, which can be thrown away afterwards.

Class diagrams - Summary

- You should not use too much energy to think about the correctness and details of the modeling language.

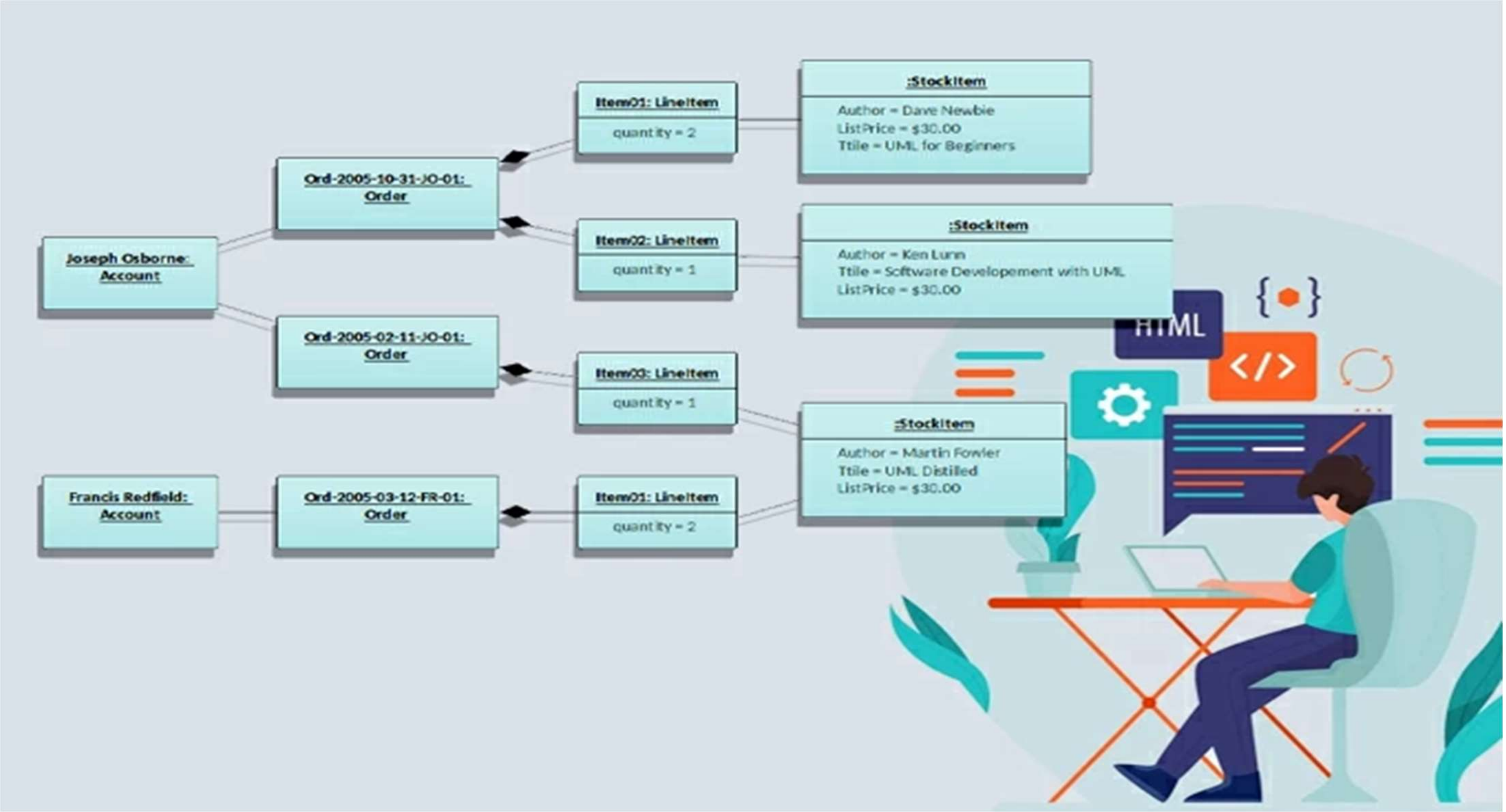
- Class diagrams should also be drawn in a suitable level of abstraction.

- For example, if you have tens of classes,

- it might not be worth describing each attribute and each method of each class;

- getting a good overview of the program structure is the most important.

Object diagram


Joseph Osborne: Account
Ord-2005-10-31-JO-01: Order
Ord-2005-02-11-JO-01: Order
Item01: LineItem
quantity = 2
:StockItem
Author = Dave Newbie
ListPrice = $30.00
Ttile = UML for Beginners
Item02: LineItem
quantity = 1
:StockItem
Author = Ken Lunn
Ttile = Software Developement with UML
ListPrice = $30.00
Item03: LineItem
quantity = 1
:StockItem
Author = Martin Fowler
Ttile = UML Distilled
ListPrice = $30.00
Francis Redfield: Account
Ord-2005-03-12-FR-01: Order
Item01: LineItem
quantity = 2
HTML
</>

What is an object diagram in UML?

- A UML object diagram represents a specific instance of a class diagram at a certain moment in time, When represented visually.

- An object diagram focuses on the attributes of a set of objects and how those objects relate to each other.

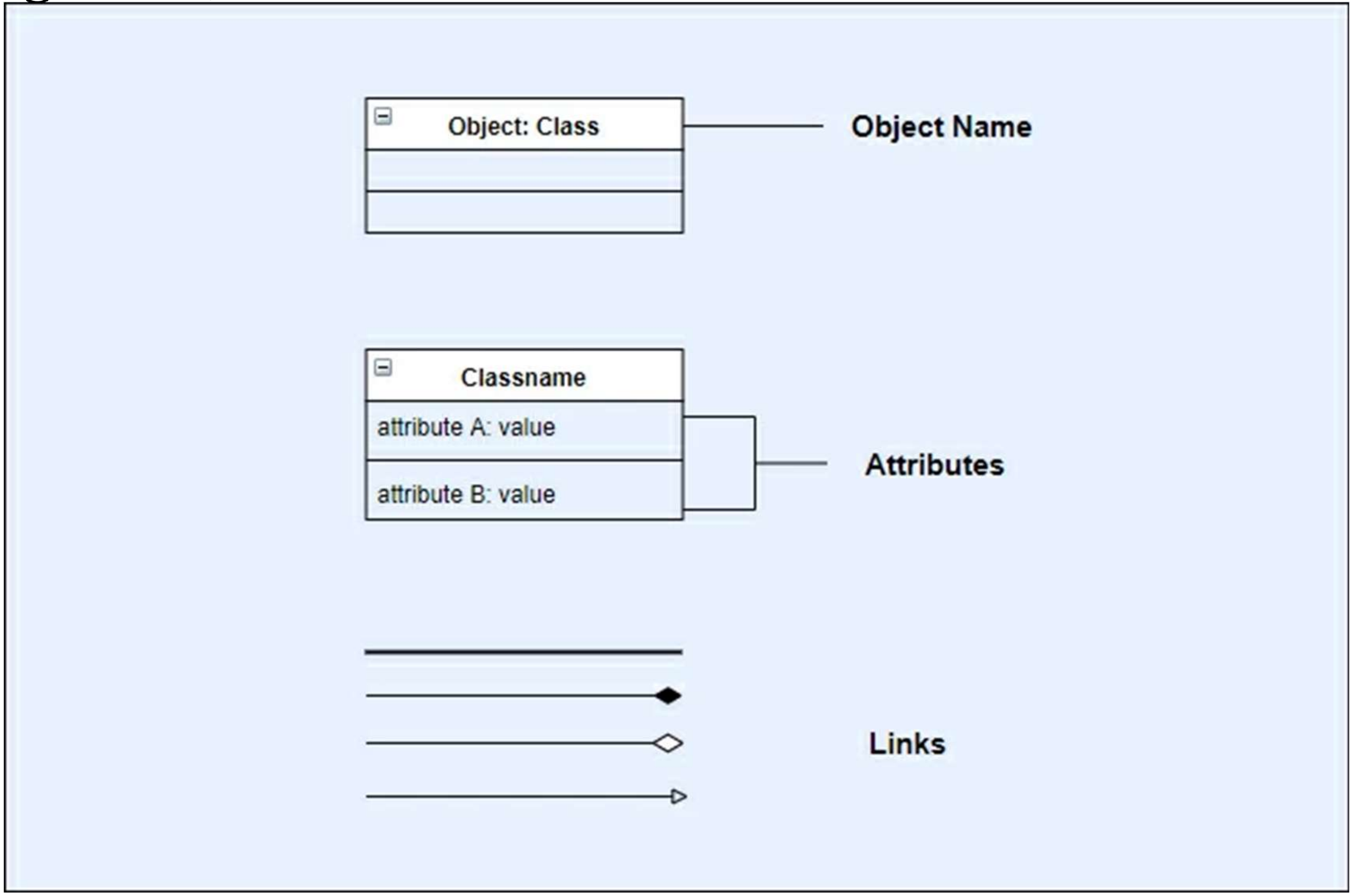
- For instance, in this object diagram below, all three bank accounts tie back to the bank itself.

- The class titles show the type of accounts (savings, checking, and credit card) that a given customer could have with this particular bank.

- The class attributes are different for each account type.

- For example, the credit card object has a credit limit, while the savings and checking accounts have interest rates.

Object diagram- Symbols and Notations


Object: Class
Object Name
Classname
attribute A: value
attribute B: value
Attributes
Links

Object diagram- Symbols and Notations

- Object Names

- Like any object in a UML diagram, a rectangle usually represents objects.
- The rectangle comes with two rows where the object name is located at the top row of the symbol.

- Attributes

- In the context of UML, attributes are the data in a class with values assigned for them.
- They have a separate compartment which is at the bottom row of the rectangular symbol.

- Links

- The purpose of links in a UML object diagram is to show relationships of the objects with each other.
- Actually, there are four links used for this diagram.
- These include straight line, composition, generalization, and aggregation.
- The most utilized link, however, is the solid, straight line.

Class vs. Object Diagram

- Class diagram and object diagram can be a bit confusing especially if you are a newbie in software engineering.
- They are closely related in terms of notations and other factors so you might find it difficult to figure out which is which.
- Therefore, we outlined the differences of each diagram by defining their purpose and showing what elements they show in a system.
- Class diagram is short software architecture of a system.
- The main purpose of this diagram is to model the objects and functionalities in a system.
- What is more, this could contain interface, packages, relationships, as well as instances including objects and links.

Object Diagram

- Object diagram on the other hand is specifically designed to display a particular instance of a class diagram at a point in time.

- Mainly, the use of this diagram is to show examples of data structures.

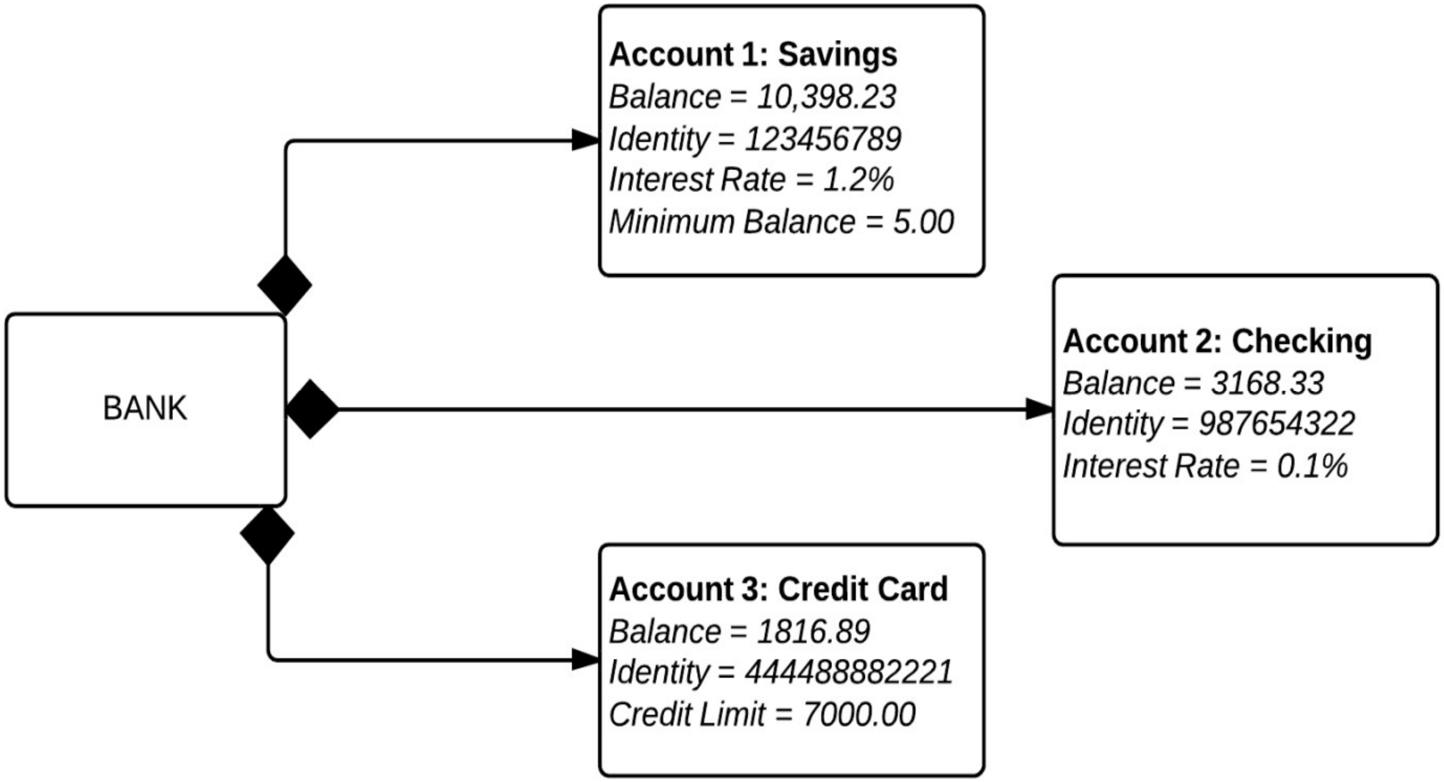
- Unlike a class diagram, it is limited to showing the data values of objects and how they interact.

- Overall, it may be said that a class diagram provides a static overview of a system.

- Whereas the UML object diagram models how objects interact with each other at some point.

- Moreover, it provides a snapshot of changes during system runtime that the class diagram does include in the display.

Object diagrams are not limited to banking use cases, however, as you can easily make an object diagram for family trees, corporate departments, or any other system with interrelated parts.



Account 1: Savings
Balance = 10,398.23
Identity = 123456789
Interest Rate = 1.2%
Minimum Balance = 5.00

BANK

Account 2: Checking
Balance = 3168.33
Identity = 987654322
Interest Rate = 0.1%

Account 3: Credit Card
Balance = 1816.89
Identity = 444488882221
Credit Limit = 7000.00

Purpose of Object Diagram

The use of object diagrams is fairly limited, mainly to show examples of data structures.

- During the analysis phase of a project.

- We might create a class diagram to describe the structure of a system and then create a set of object diagrams, as test cases to verify the accuracy and completeness of the class diagram.

- Before you create a class diagram, you might create an object diagram to discover facts about specific model elements and their links.

- To illustrate specific examples of the classifiers that are required.

Purpose of Object Diagrams

- The purpose of a diagram should be understood clearly to implement it practically.
- The purposes of object diagrams are similar to class diagrams.
- The difference is that a class diagram represents an abstract model consisting of classes and their relationships.
- However, an object diagram represents an instance at a particular moment, which is concrete in nature.
- It means the object diagram is closer to the actual system behavior.
- The purpose is to capture the static view of a system at a particular moment.

Purpose of Object Diagrams

- The purpose of the object diagram can be summarized as −
- Forward and reverse engineering.
- Object relationships of a system
- Static view of an interaction.
- Understand object behaviour and their relationship from practical perspective

How to Draw an Object Diagram?

- We have already discussed that an object diagram is an instance of a class diagram.

- It implies that an object diagram consists of instances of things used in a class diagram.

- So both diagrams are made of same basic elements but in different form.

- In class diagram elements are in abstract form to represent the blue print and in object diagram the elements are in concrete form to represent the real world object.

How to Draw an Object Diagram?

- To capture a particular system, numbers of class diagrams are limited.

- However, if we consider object diagrams then we can have unlimited number of instances, which are unique in nature.

- Only those instances are considered, which have an impact on the system.

How to Draw an Object Diagram?

- it is clear that a single object diagram cannot capture all the necessary instances or rather cannot specify all the objects of a system. Hence, the solution is −
- First, analyze the system and decide which instances have important data and association.
- Second, consider only those instances, which will cover the functionality.
- Third, make some optimization as the number of instances are unlimited.

How to Draw an Object Diagram?

- Before drawing an object diagram, the following things should be remembered and understood clearly –

- Object diagrams consist of objects.

- The link in object diagram is used to connect objects.

- Objects and links are the two elements used to construct an object diagram.

The following things are to be decided before starting the construction of the diagram −

- The object diagram should have a meaningful name to indicate its purpose.

- The most important elements are to be identified.

- The association among objects should be clarified.

- Values of different elements need to be captured to include in the object diagram.

- Add proper notes at points where more clarity is required.

Order management system

- The following diagram is an instance of the system at a particular time of purchase.

- It has the following objects.

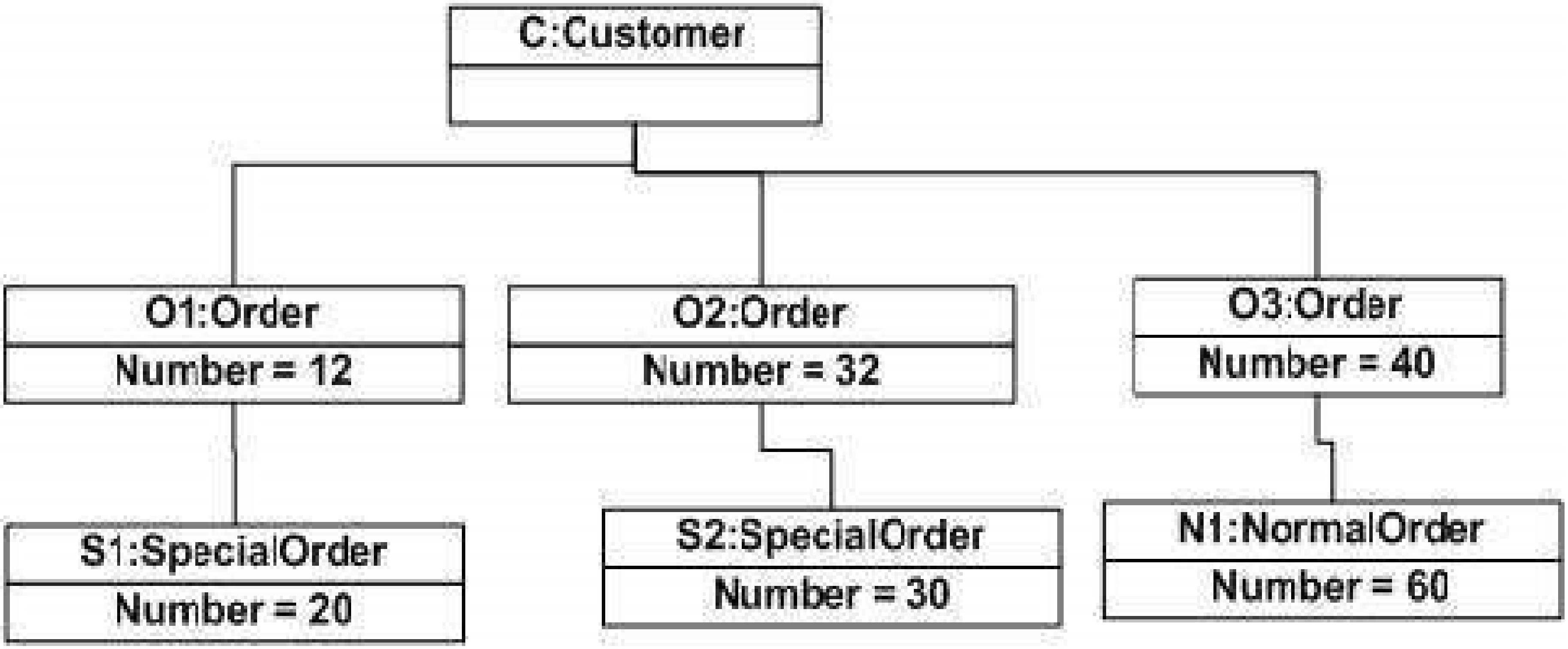
- Customer

- Order

- SpecialOrder

- NormalOrder

Object diagram of an order management system



C:Customer

O1:Order

Number = 12

O2:Order

Number = 32

O3:Order

Number = 40

S1:SpecialOrder

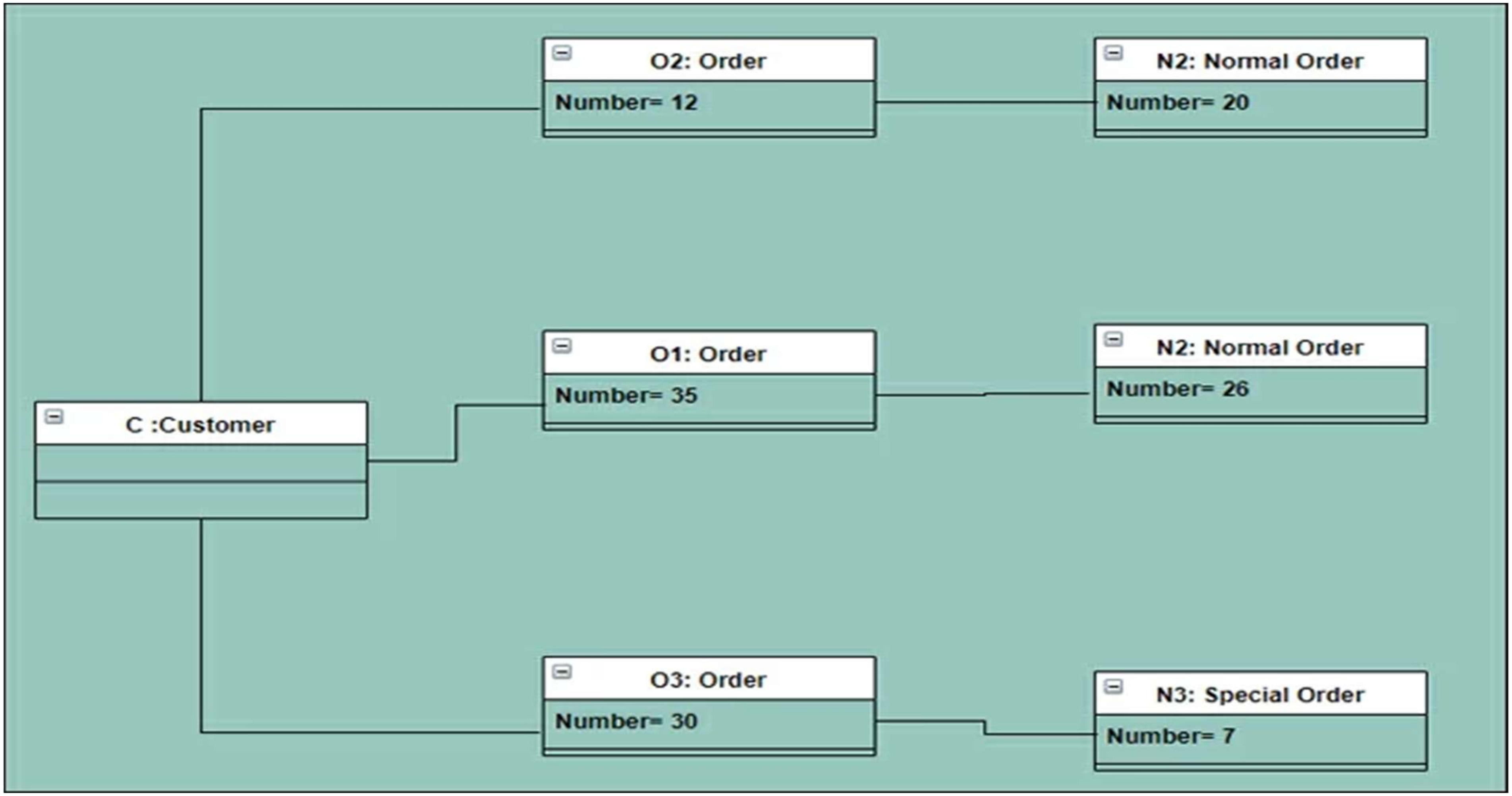
Number = 20

S2:SpecialOrder

Number = 30

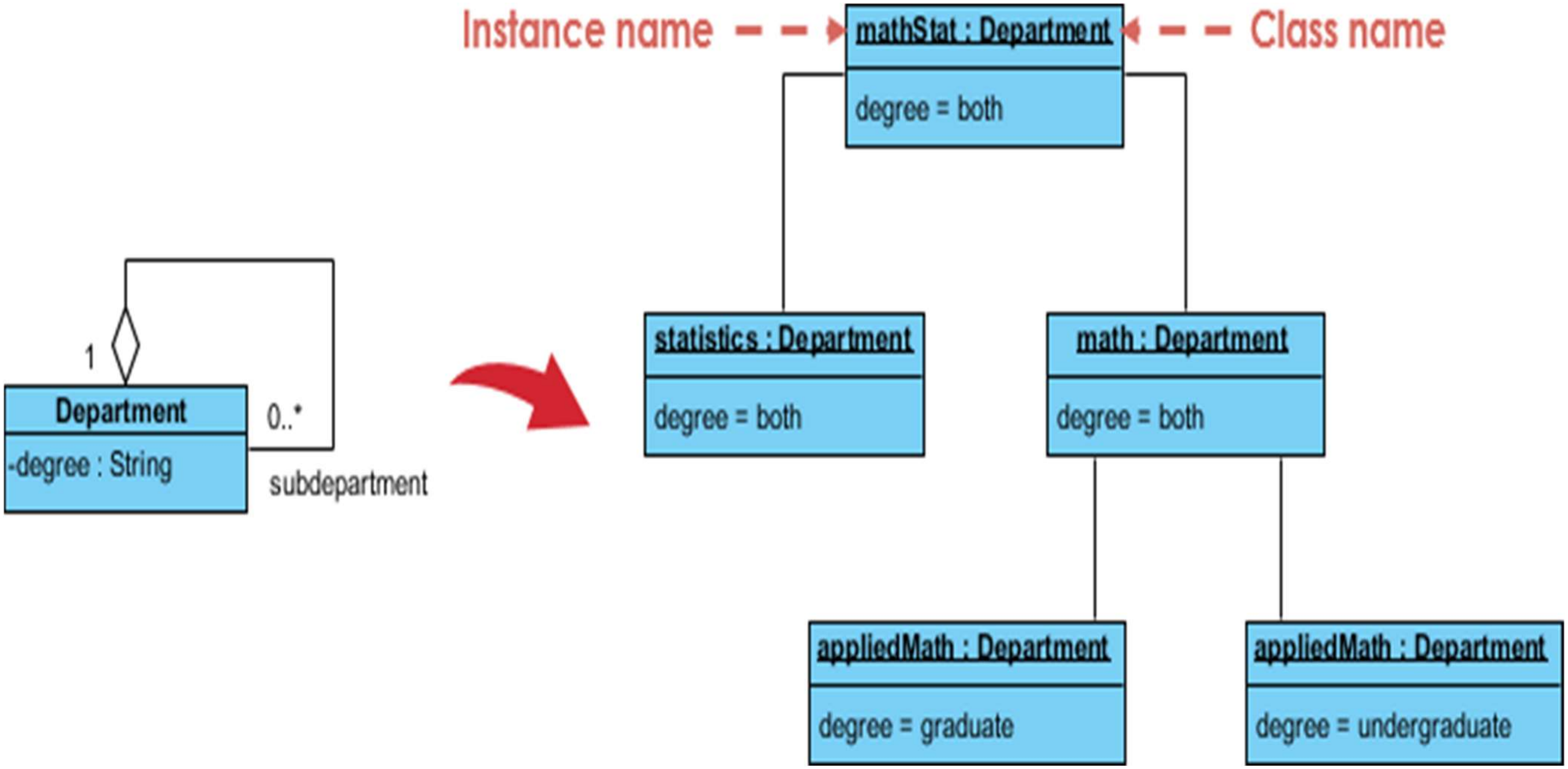
N1:NormalOrder

Number = 60

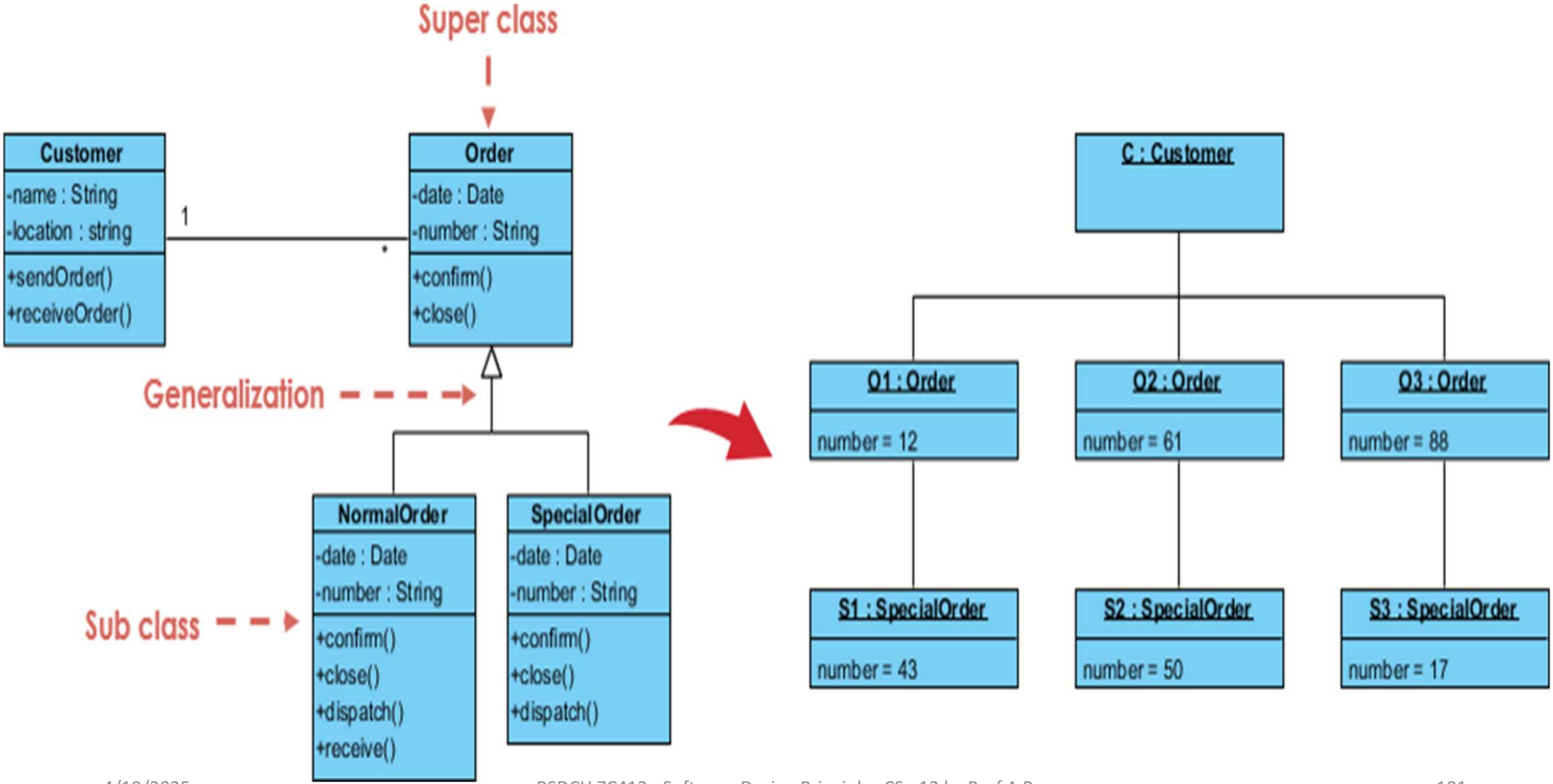

O2: Order
Number= 12
N2: Normal Order
Number= 20
O1: Order
Number= 35
N2: Normal Order
Number= 26
C :Customer
O3: Order
Number= 30
N3: Special Order
Number= 7

Object Diagram at a Glance

- An object diagram shows this relation between the instantiated classes and the defined class, and the relation between these objects in the system.
- They are be useful to explain smaller portions of your system, when your system class diagram is very complex, and also sometimes modeling recursive relationship in diagram.
- The best way to illustrate what an object diagram look like is to show the object diagram derived from the corresponding class diagram.
- The following Order Management System shows their relationships.
- This small class diagram shows that a university Department can contain lots of other Departments and the object diagram below instantiates the class diagram, replacing it by a concrete example.

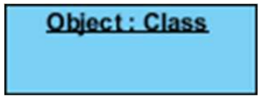

Instance name
Class name
mathStat : Department
degree = both
1
Department
-degree : String
0..*
subdepartment
statistics : Department
degree = both
math : Department
degree = both
appliedMath : Department
degree = graduate
appliedMath : Department
degree = undergraduate

Class to Object Diagram Example - Order System

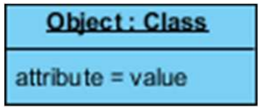

Super class
Customer
-name : String
-location : string
+sendOrder()
+receiveOrder()
1
*
Order
-date : Date
-number : String
+confirm()
+close()
Generalization
Sub class
NormalOrder
-date : Date
-number : String
+confirm()
+close()
+dispatch()
+receive()
SpecialOrder
-date : Date
-number : String
+confirm()
+close()
+dispatch()
C : Customer
O1 : Order
number = 12
O2 : Order
number = 61
O3 : Order
number = 88
S1 : SpecialOrder
number = 43
S2 : SpecialOrder
number = 50
S3 : SpecialOrder
number = 17

Basic Object Diagram Symbols and Notations

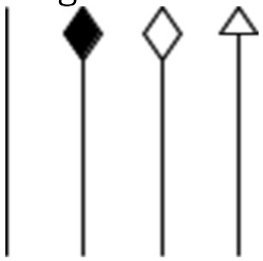
- Object Names:



- Every object is actually symbolized like a rectangle, that offers the name from the object and its class underlined as well as divided with a colon.
- Object Attributes:



- Similar to classes, you are able to list object attributes inside a separate compartment. However, unlike classes, object attributes should have values assigned for them.
- Links:



- Links tend to be instances associated with associations. You can draw a link while using the lines utilized in class diagrams.

Class Diagram vs. Object Diagram

- In UML, object diagrams provide a snapshot of the instances in a system and the relationships between the instances.

- By instantiating the model elements in a class diagram, we can explore the behavior of a system at a point in time.

- An object diagram is a UML structural diagram that shows the instances of the classifiers in models.

- Object diagrams use notation that is similar to that used in class diagrams.

- Class diagrams show the actual classifiers and their relationships in a system

- Object diagrams show specific instances of those classifiers and the links between those instances at a point in time.

- You can create object diagrams by instantiating the classifiers in class, deployment, component, and use-case diagrams.

Steps for Modeling Object Structures

- A communication diagram (called collaboration in previous version of UML) without messages is also known as an object diagram, and the relationships between objects are called links.

- An object diagram must be a valid instantiation of a static class diagram.

- Objects must have classes and links between objects must be instances of associations between classes.

- This can be used as a quick consistency check.

Steps for Modeling Object Structures

- To do this we can develop an object diagram by using the following steps:

- Identify the **mechanism** you'd like to model.

- A mechanism represents some **functions** or **behaviors** of the part of the system you are **modeling that results from the interaction of a society of classes, interfaces, and other things**.

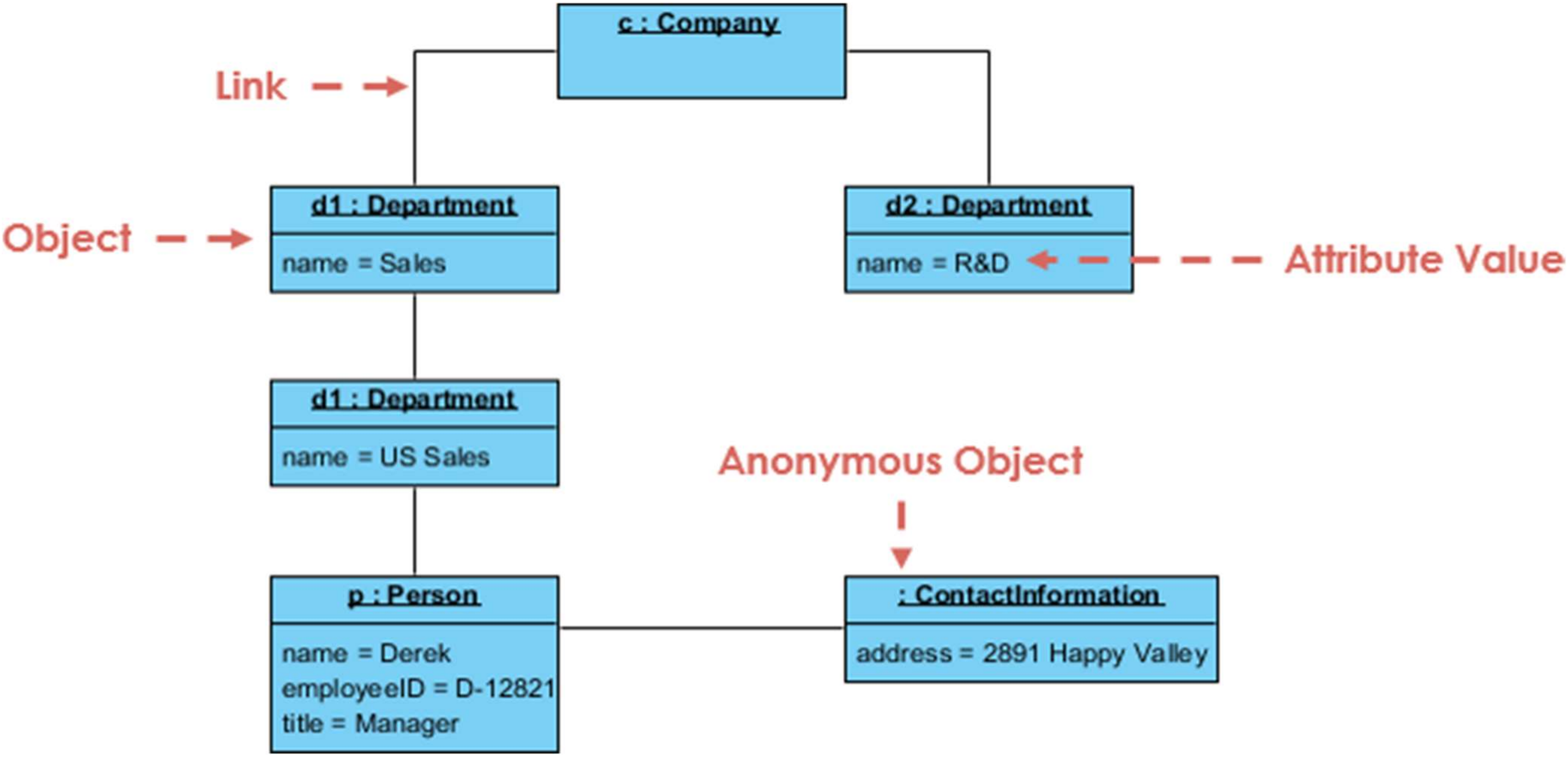
- For each mechanism, identify the classes, interfaces, and other elements that participate in this collaboration; identify the relationships among these things, as well.

- Consider one scenario that walks through this mechanism.

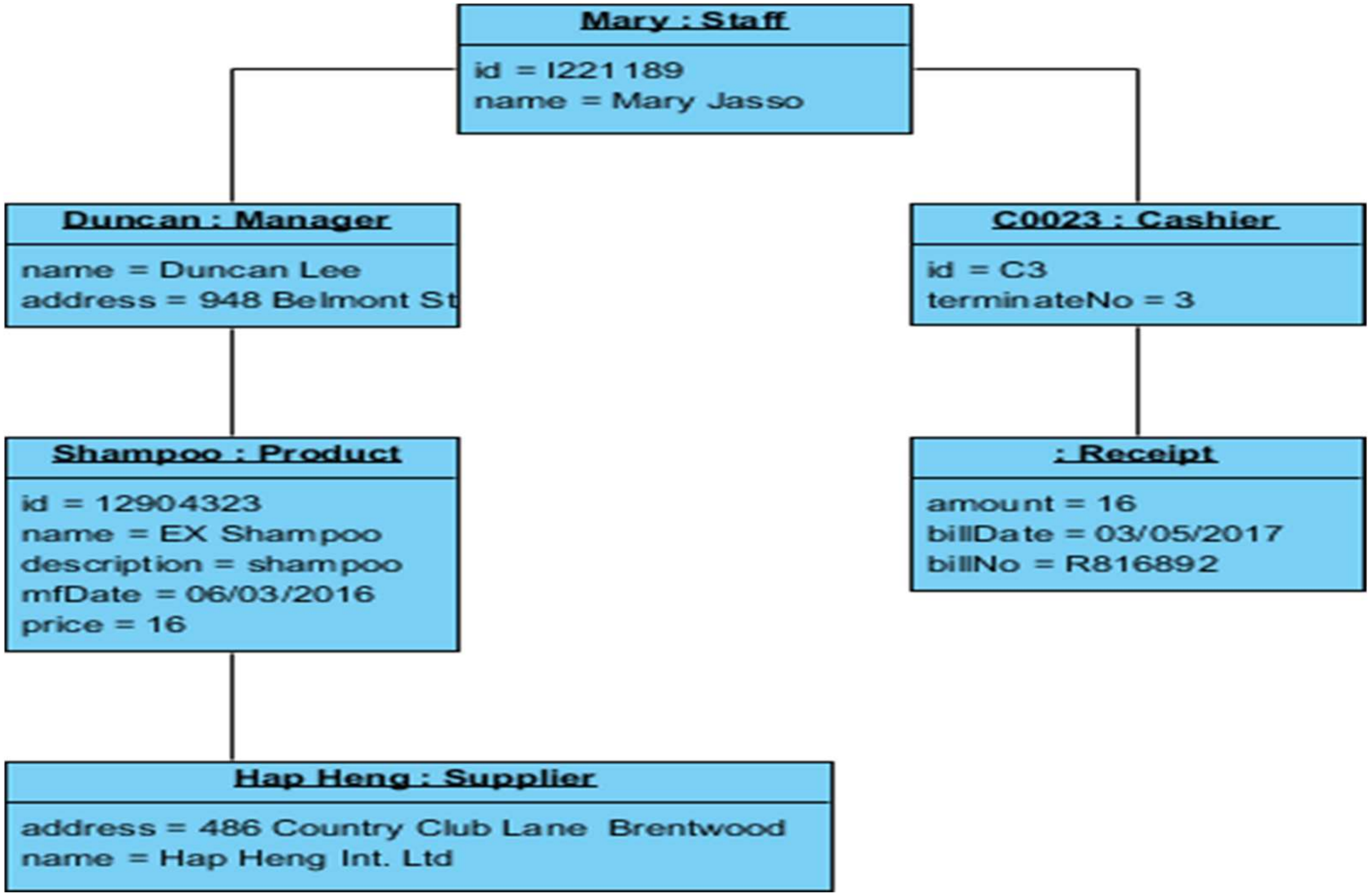
- Freeze that scenario at a moment in time, and render each object that participates in the mechanism.

- Expose the state and attribute values of each such object, as necessary, to understand the scenario.

Object Diagram Example I - Company Structure


c : Company
Link
Object
d1 : Department
name = Sales
d2 : Department
name = R&D
Attribute Value
d1 : Department
name = US Sales
Anonymous Object
p : Person
name = Derek
employeeID = D-12821
title = Manager
: ContactInformation
address = 2891 Happy Valley

Object Diagram Example II - POS



Mary : Staff
id = I221189
name = Mary Jasso

Duncan : Manager
name = Duncan Lee
address = 948 Belmont St

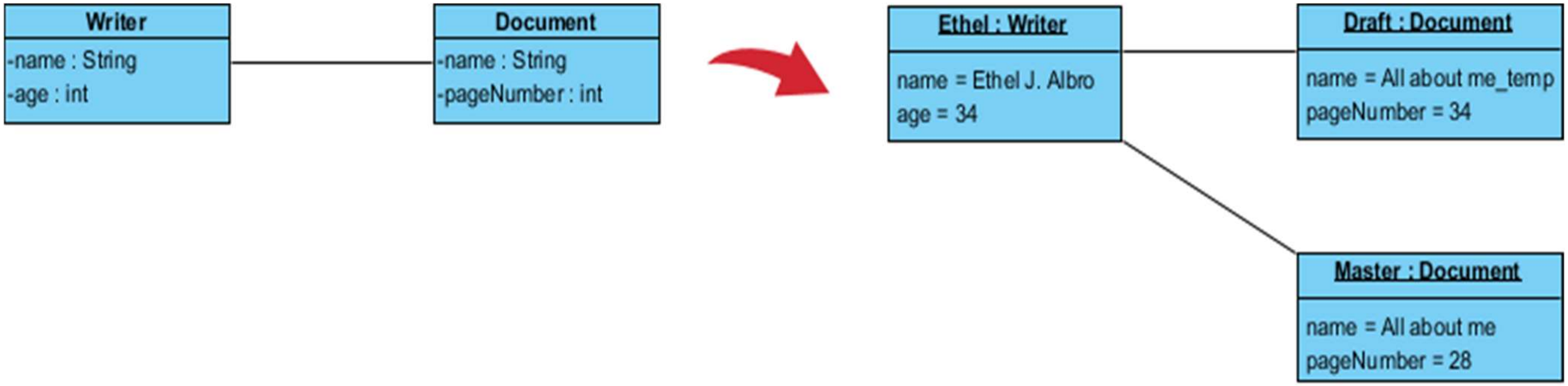
C0023 : Cashier
id = C3
terminateNo = 3

Shampoo : Product
id = 12904323
name = EX Shampoo
description = shampoo
mfDate = 06/03/2016
price = 16

: Receipt
amount = 16
billDate = 03/05/2017
billNo = R816892

Hap Heng : Supplier
address = 486 Country Club Lane Brentwood
name = Hap Heng Int. Ltd

Object Diagram Example III - Writer



Writer

-name : String
-age : int

Document

-name : String
-pageNumber : int

Ethel : Writer

name = Ethel J. Albro
age = 34

Draft : Document

name = All about me_temp
pageNumber = 34

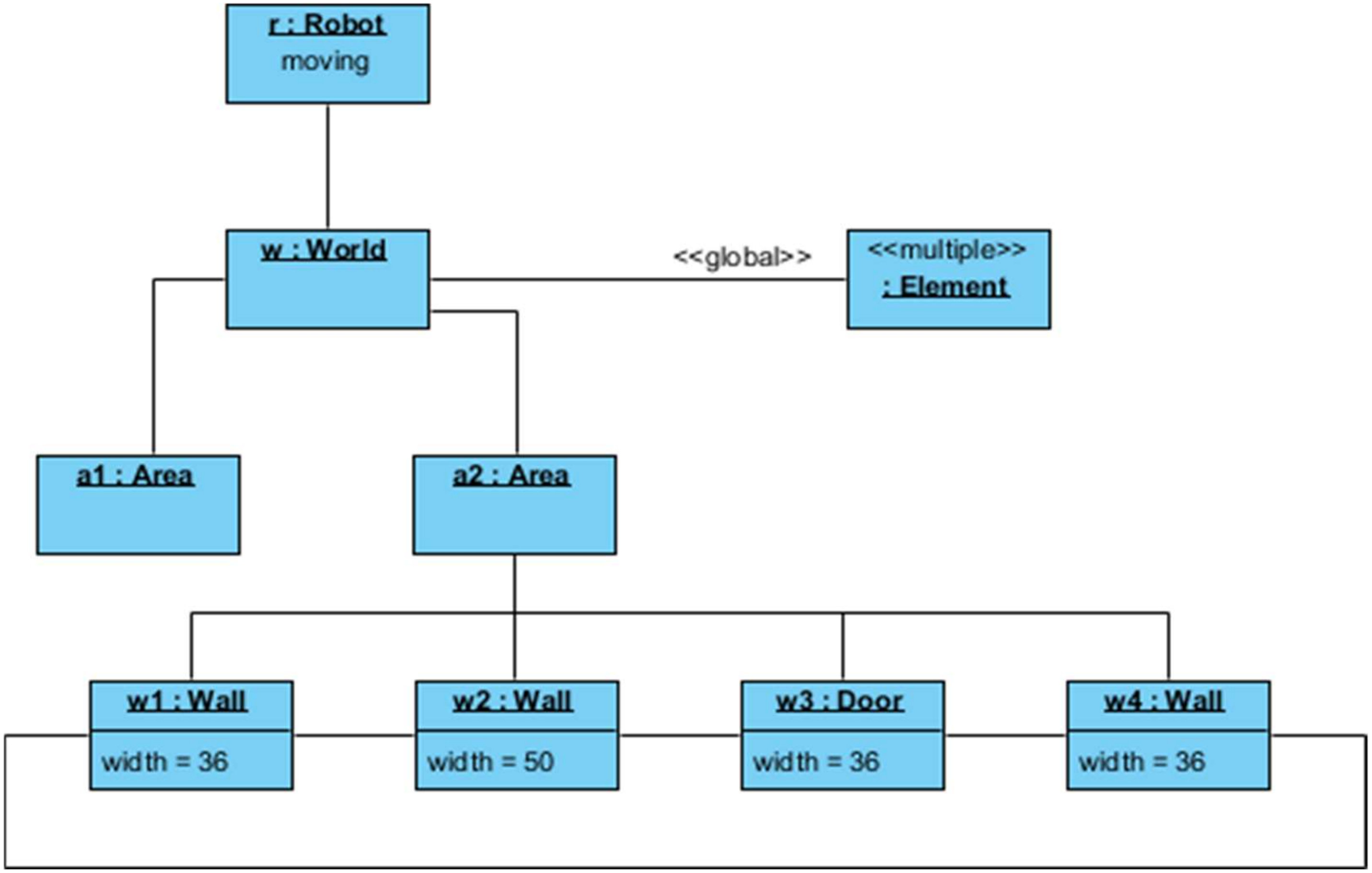
Master : Document

name = All about me
pageNumber = 28

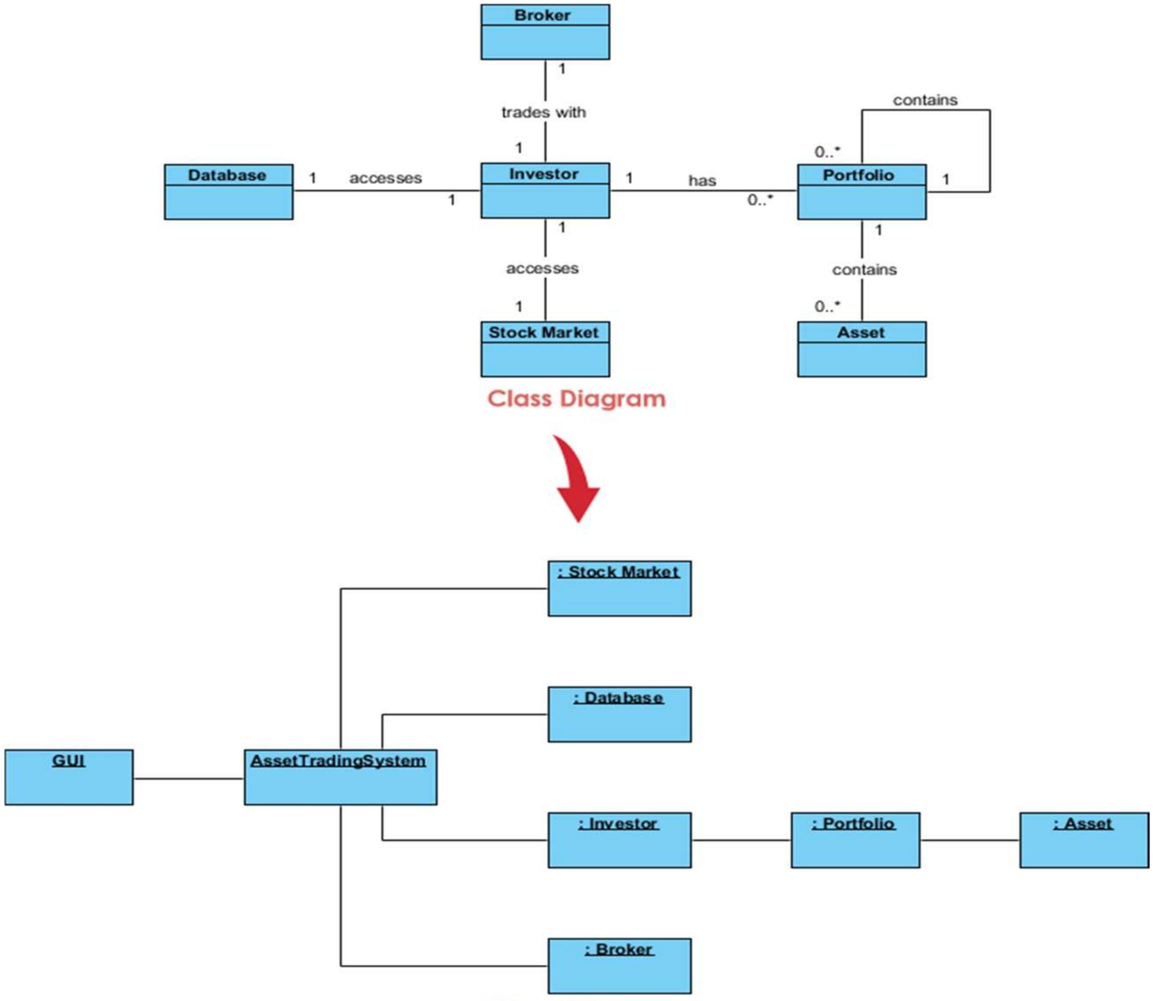
Class Diagram

Object Diagram

Object Diagram Example IV - Robot Moving Behavior

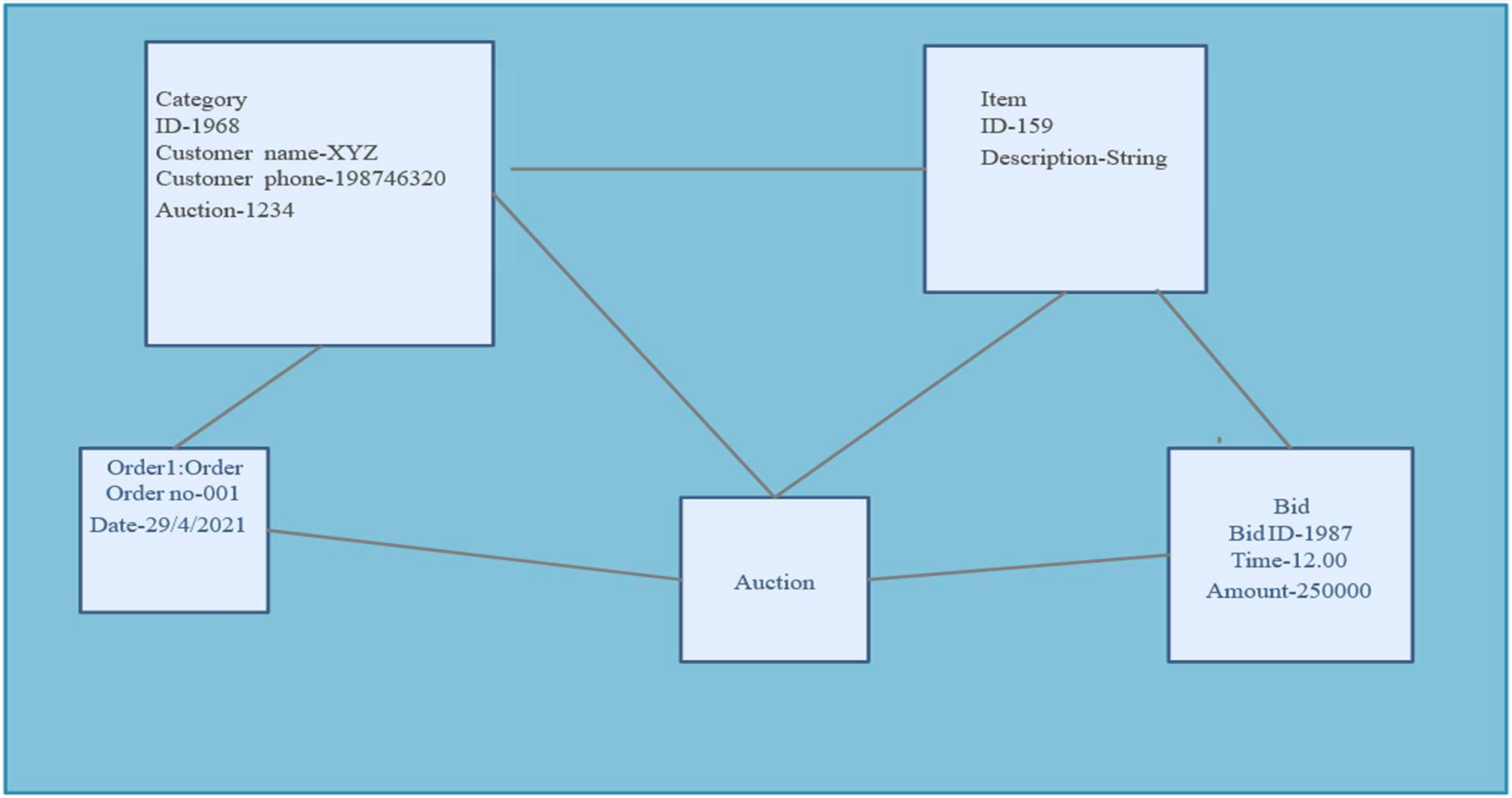

r : Robot
moving
w : World
<<global>>
<<multiple>>
: Element
a1 : Area
a2 : Area
w1 : Wall
width = 36
w2 : Wall
width = 50
w3 : Door
width = 36
w4 : Wall
width = 36

Object Diagram Example V - Deriving an Object Structure Similar to Communication Diagram


Broker
1
trades with
1
Database
1
accesses
1
Investor
1
has
0..*
Portfolio
contains
0..*
1
1
accesses
1
Stock Market
1
contains
0..*
Asset
: Stock Market
: Database
GUI
AssetTradingSystem
: Investor
: Portfolio
: Asset
: Broker

Class Diagram

Object Diagram

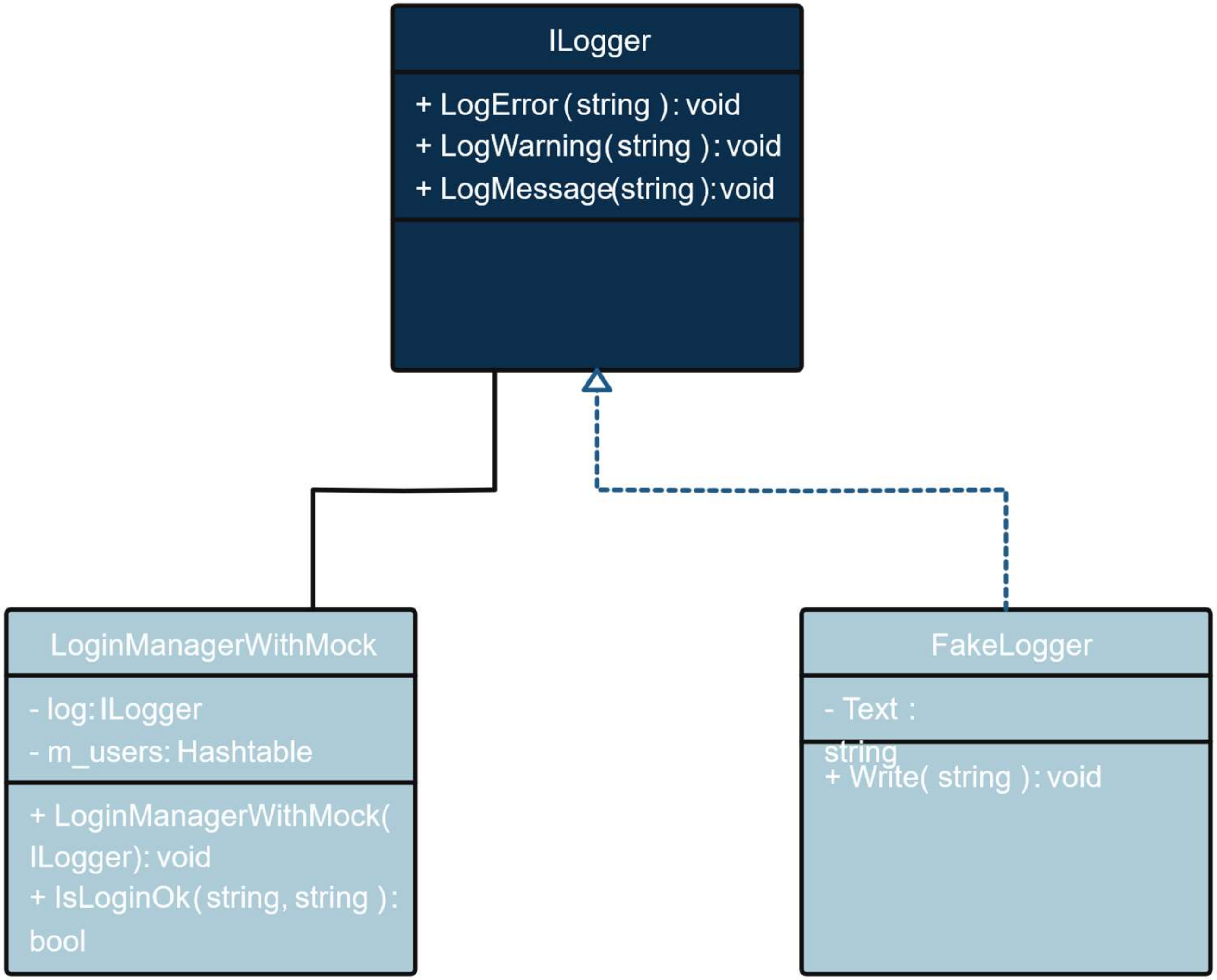

Category
ID-1968
Customer name-XYZ
Customer phone-198746320
Auction-1234

Item
ID-159
Description-String

Order1:Order
Order no-001
Date-29/4/2021

Auction

Bid
BidID-1987
Time-12.00
Amount-250000



ILogger

+ LogError (string) : void
+ LogWarning (string) : void
+ LogMessage(string) : void

LoginManagerWithMock

- log: ILogger
- m_users: Hashtable

+ LoginManagerWithMock(ILogger): void
+ IsLoginOk (string, string) : bool

FakeLogger

- Text : string

+ Write(string) : void

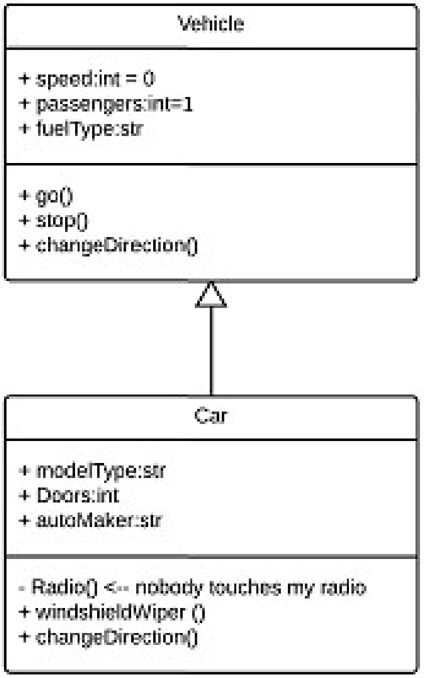
- Object diagram elements

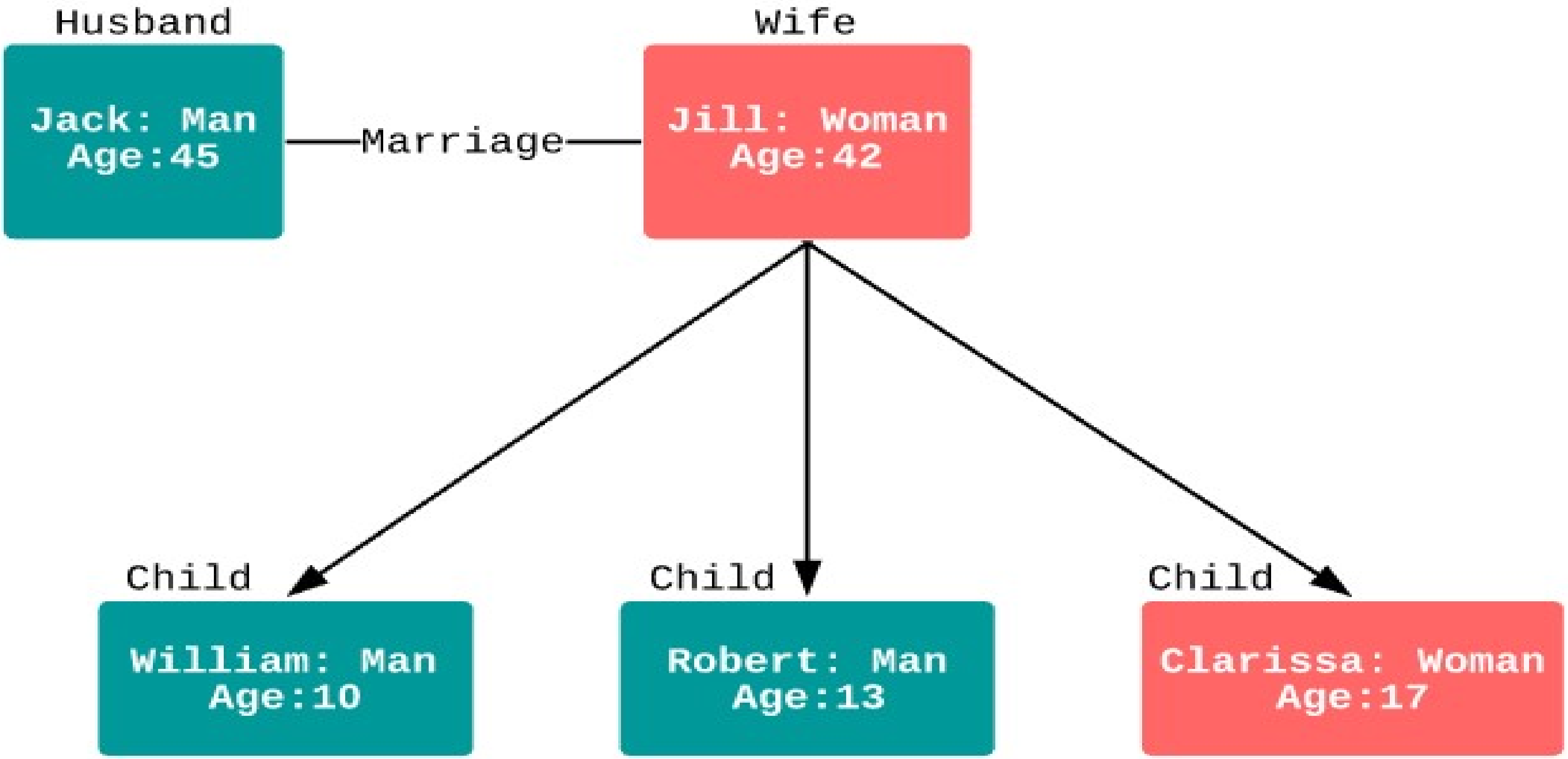
- Object diagrams are simple to create: they're made from objects, represented by rectangles, linked together with lines.
- Take a look at the major elements of an object diagram.

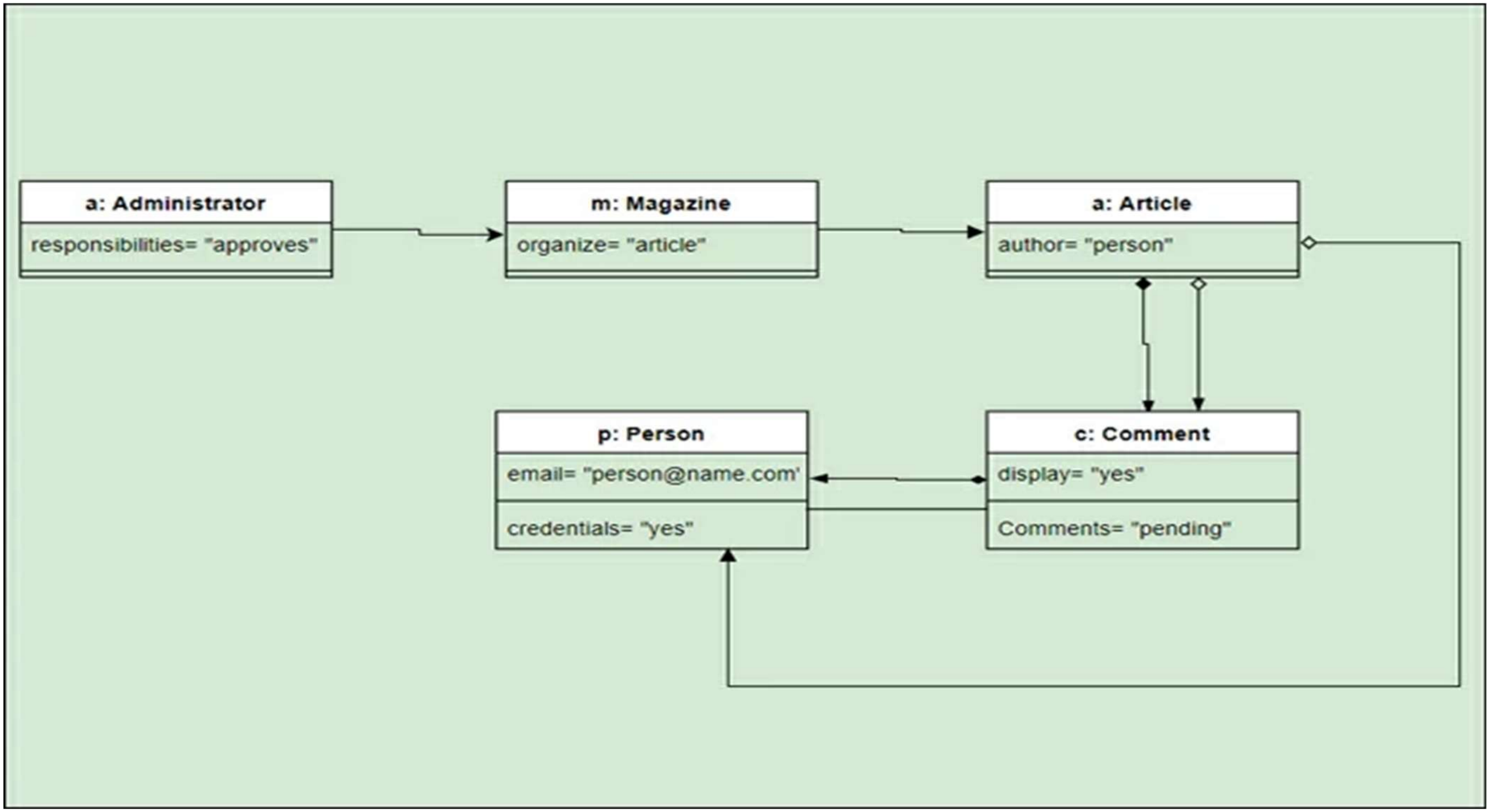
- Objects
- Objects are instances of a class.
- For example, if "car" is a class, a 2007 Nissan Altima is an object of a class.

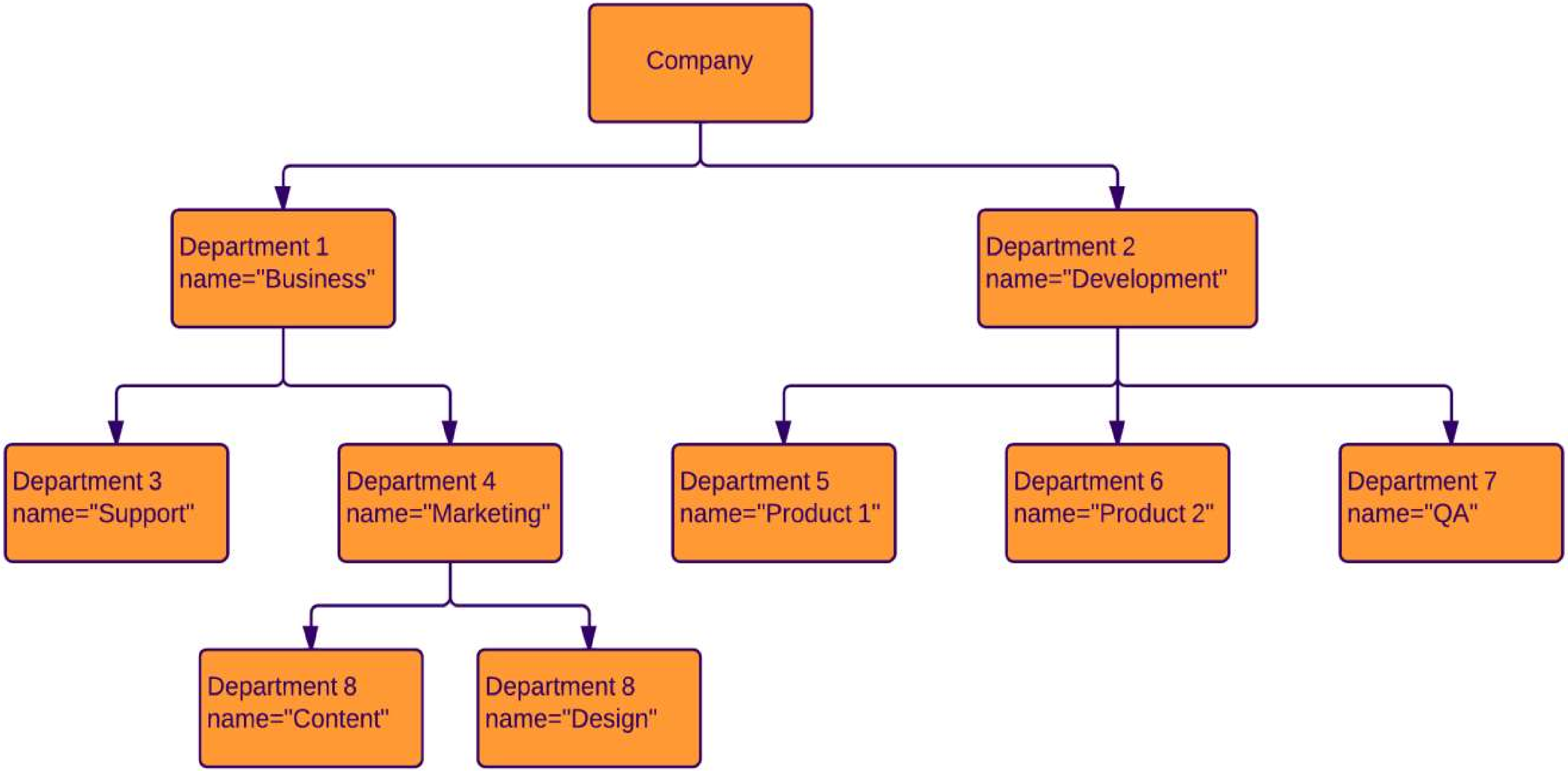
- Class titles

- Class titles are the specific attributes of a given class. In the family tree object diagram, class titles include the name, gender, and age of the family members.
- You can list class titles as items on the object or even in the properties of the object itself (such as color).


Vehicle
+ speed:int = 0
+ passengers:int=1
+ fuelType:str
+ go()
+ stop()
+ changeDirection()
Car
+ modelType:str
+ Doors:int
+ autoMaker:str
- Radio() <-- nobody touches my radio
+ windshieldWiper ()
+ changeDirection()


Husband
Jack: Man
Age:45
Marriage
Wife
Jill: Woman
Age:42
Child
William: Man
Age:10
Child
Robert: Man
Age:13
Child
Clarissa: Woman
Age:17


a: Administrator
responsibilities= "approves"
m: Magazine
organize= "article"
a: Article
author= "person"
p: Person
email= "person@name.com'
credentials= "yes"
c: Comment
display= "yes"
Comments= "pending"



Company

Department 1
name="Business"

Department 2
name="Development"

Department 3
name="Support"

Department 4
name="Marketing"

Department 5
name="Product 1"

Department 6
name="Product 2"

Department 7
name="QA"

Department 8
name="Content"

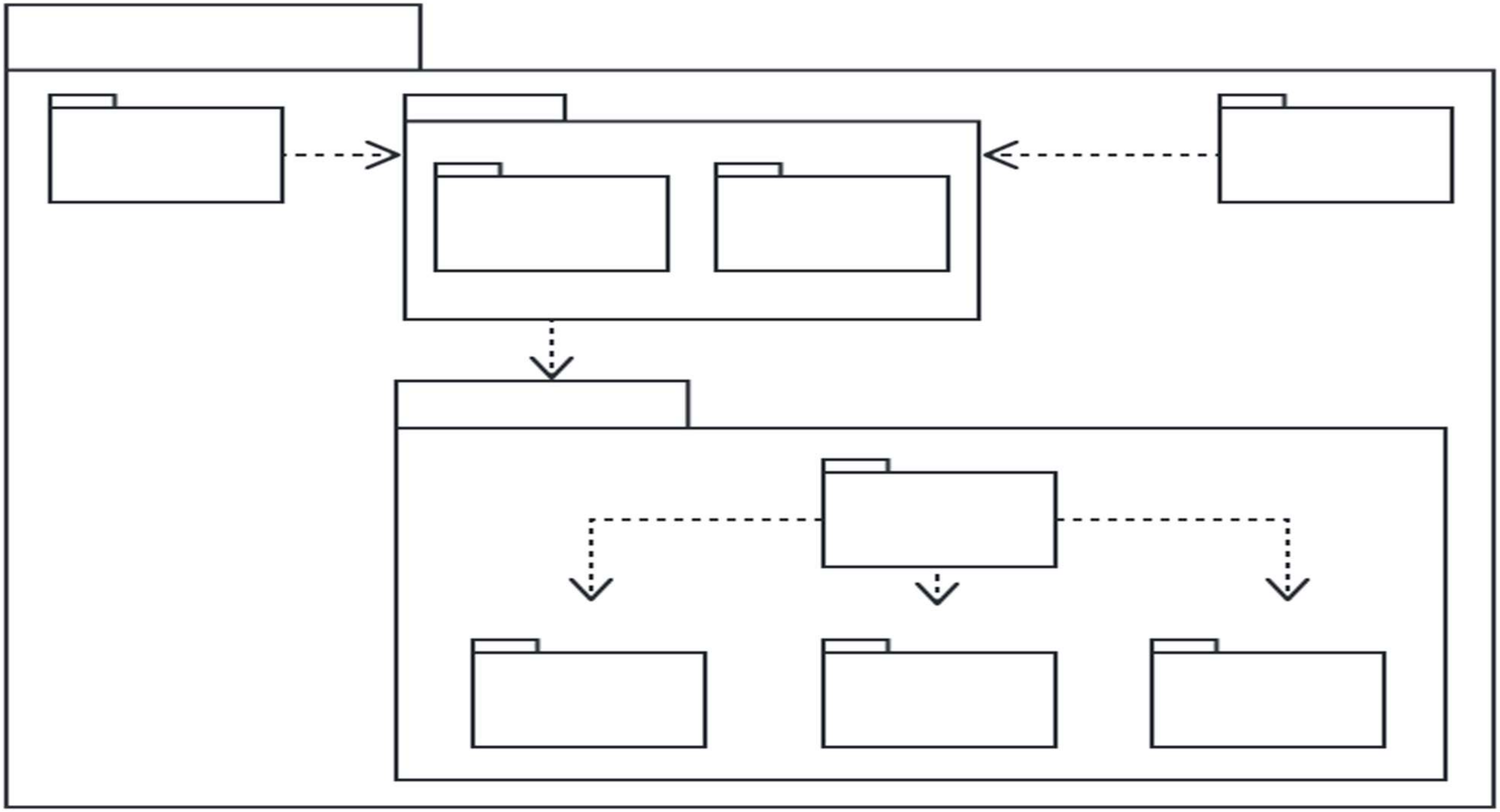
Department 8
name="Design"

Object diagram applications

- A developer will find object diagrams useful in many instances.
- These include:
- Examining a specific iteration of a general system.
- Getting a high-level overview of the system you will develop.
- Testing a class diagram we’ve created for the overall structure of the system, using object diagrams for specific use cases

Package Diagram

Package Diagram



What is a package diagram?

- Package diagrams are structural diagrams used to show the organization and arrangement of various model elements in the form of packages.

- A package is a grouping of related UML elements, such as diagrams, documents, classes, or even other packages.

- Each element is nested within the package, which is depicted as a file folder within the diagram, then arranged hierarchically within the diagram.

- Package diagrams are most commonly used to provide a visual organization of the layered architecture within any UML classifier, such as a software system.

Purpose of Package Diagrams

- Package diagrams are used to structure high level system elements.

- Packages are used for organizing large system which contains diagrams, documents and other key deliverables.

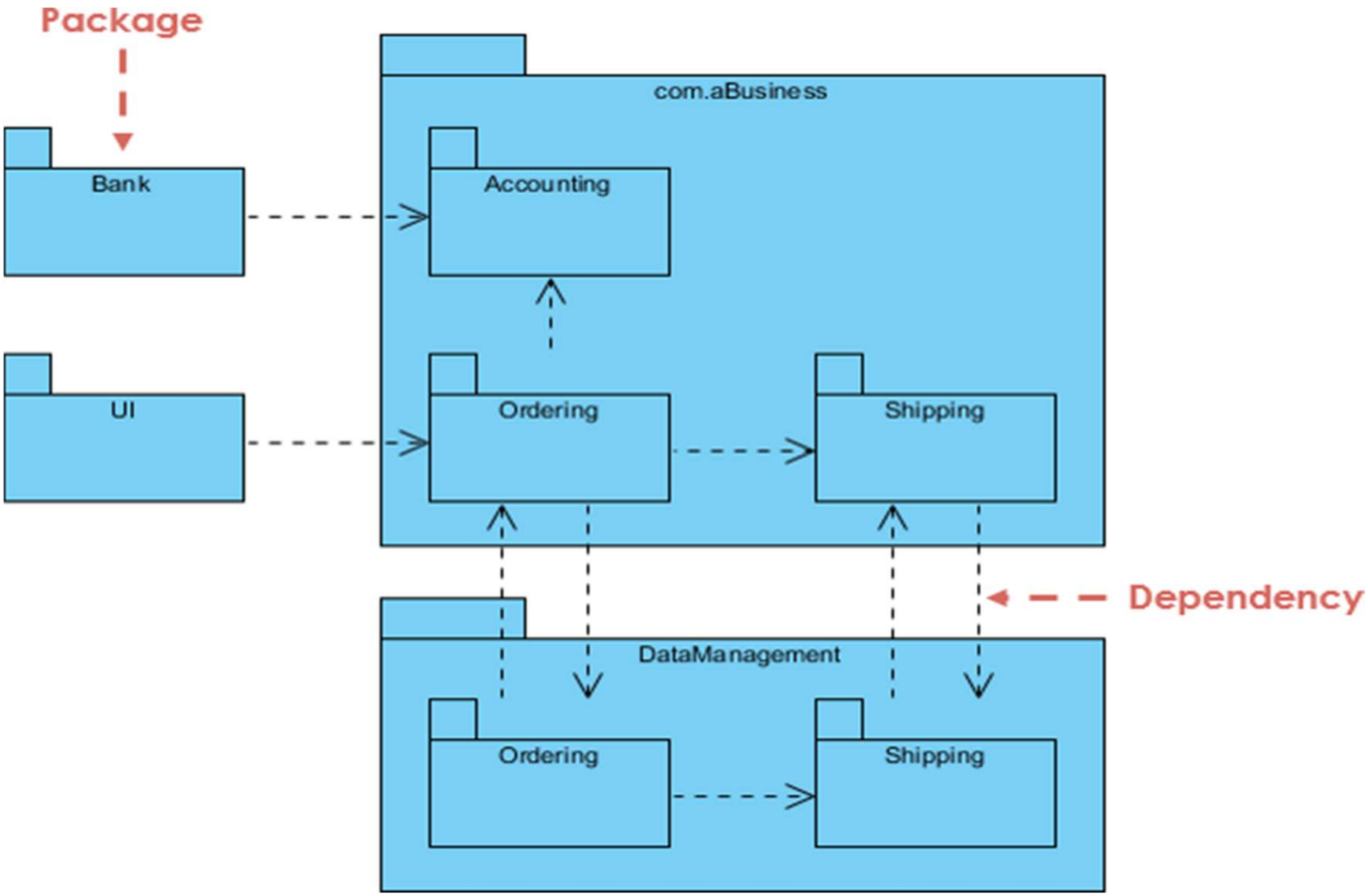
- Package Diagram can be used to simplify complex class diagrams, it can group classes into packages.

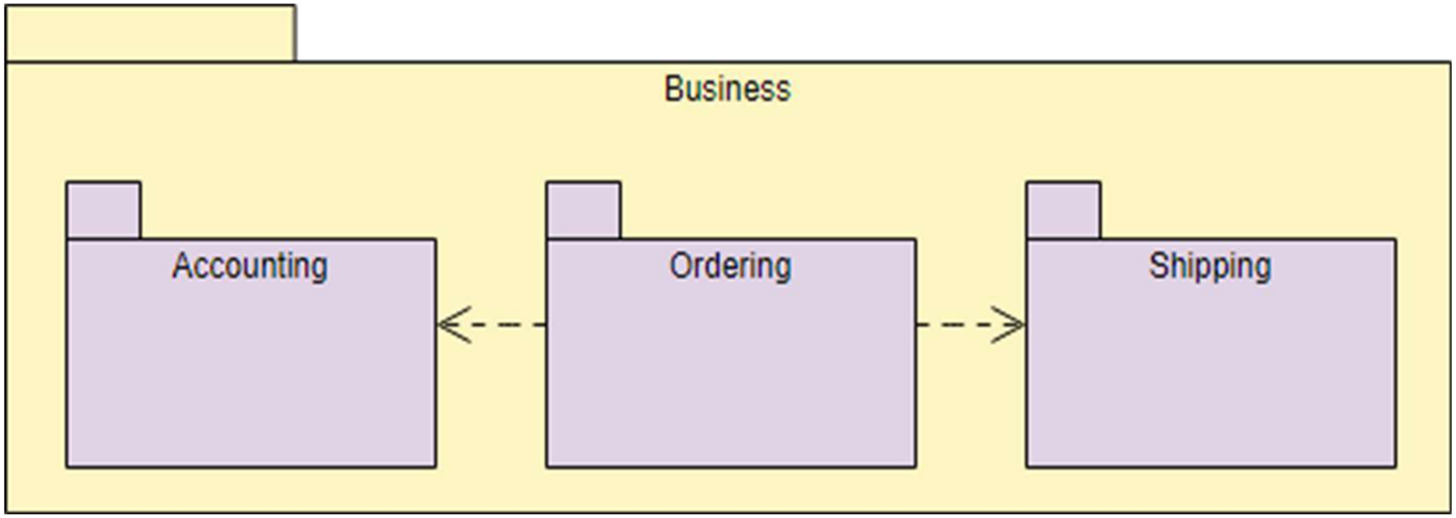
- A package is a collection of logically related UML elements.

- Packages are depicted as file folders and can be used on any of the UML diagrams.

Package Diagram at a Glance

- Package diagram is used to simplify complex class diagrams, you can group classes into packages.
- A package is a collection of logically related UML elements.
- The diagram below is a business model in which the classes are grouped into packages:
- Packages appear as rectangles with small tabs at the top.
- The package name is on the tab or inside the rectangle.
- The dotted arrows are dependencies.
- One package depends on another if changes in the other could possibly force changes in the first.


Package
Bank
UI
com.aBusiness
Accounting
Ordering
Shipping
Dependency
DataManagement
Ordering
Shipping


Business
Accounting
Ordering
Shipping

Packages

- As the number of classes implemented for the program grows, remembering all the functionality and methods becomes more difficult.

- What helps is naming the classes in a sensible manner and planning them so that each class has one clear responsibility.

- In addition to these measures, it's wise to divide the classes into packages.

- Classes in one package might share functionality, purpose, or some other logical property.

Packages

- Packages are practically directories in which the source code files are organised.

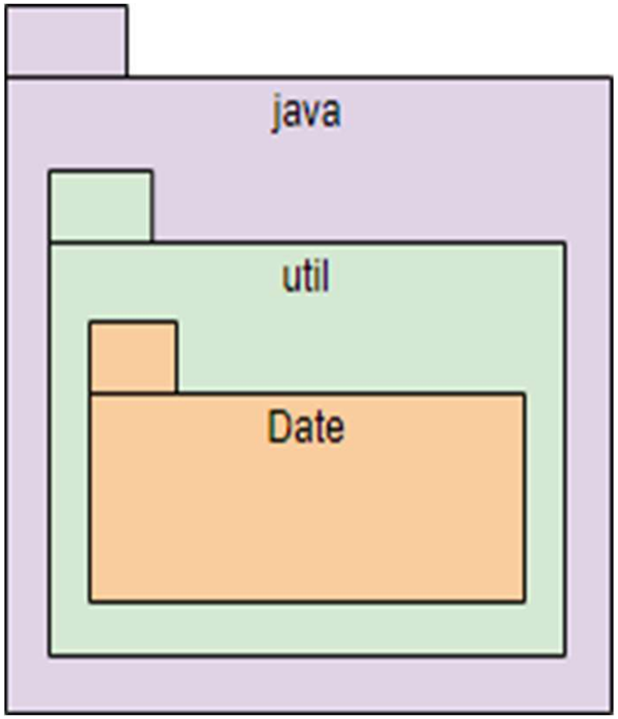
- IDEs offer existing tools for package management.

- Up until this point, we have only created classes and interfaces in the default package of the Source Packages folder of the project.

- You can create a new package in NetBeans by right-clicking on the Source Packages section (which contains the project's packages), and then selecting *New -> Java Package...*.

Nested and Hierarchical Packages

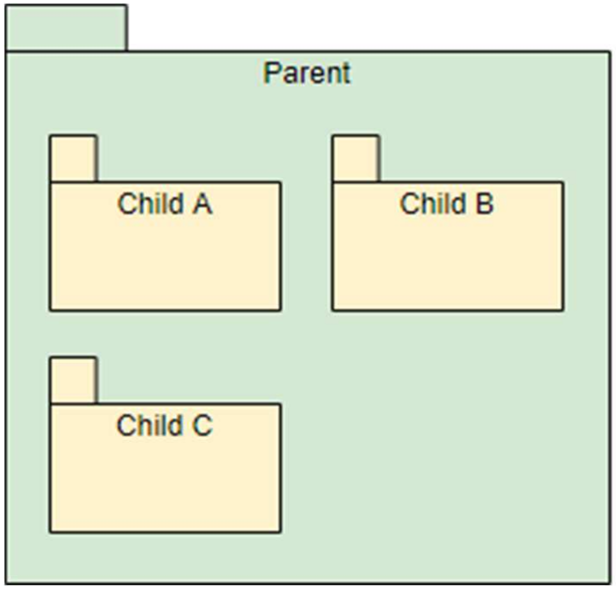
- A package can be represented as a hierarchical structure with nested packages.
- Atomic module for nested package are usually class diagrams.
- The figure below gives an example of package diagram that consists of several nested packages.

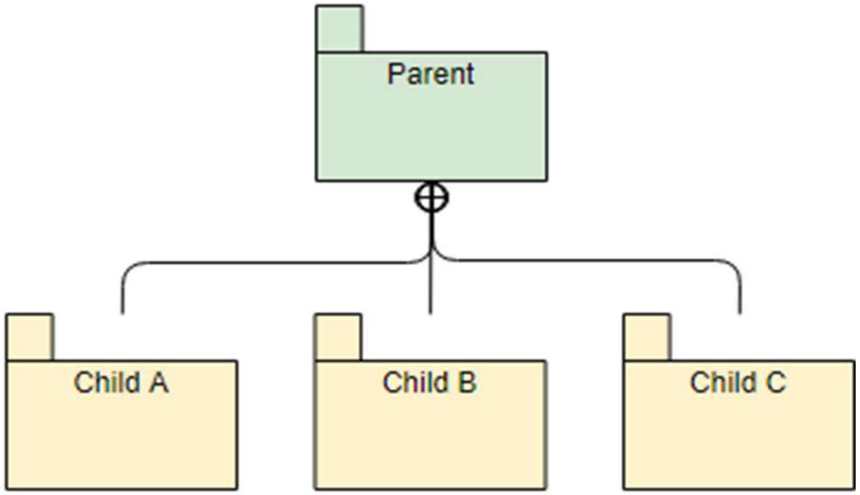


Package Containment

- Packages are shown in static diagrams

- Two equivalent ways to show containment:


Parent
Child A
Child B
Child C


Parent
Child A
Child B
Child C

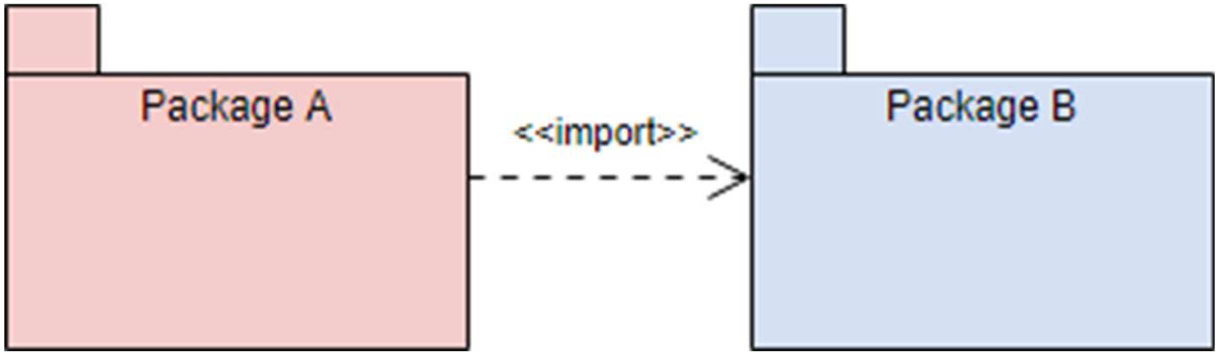
Dependency

- There are two sub-types involved in dependency.

- They are <<access>> & <<import>>.

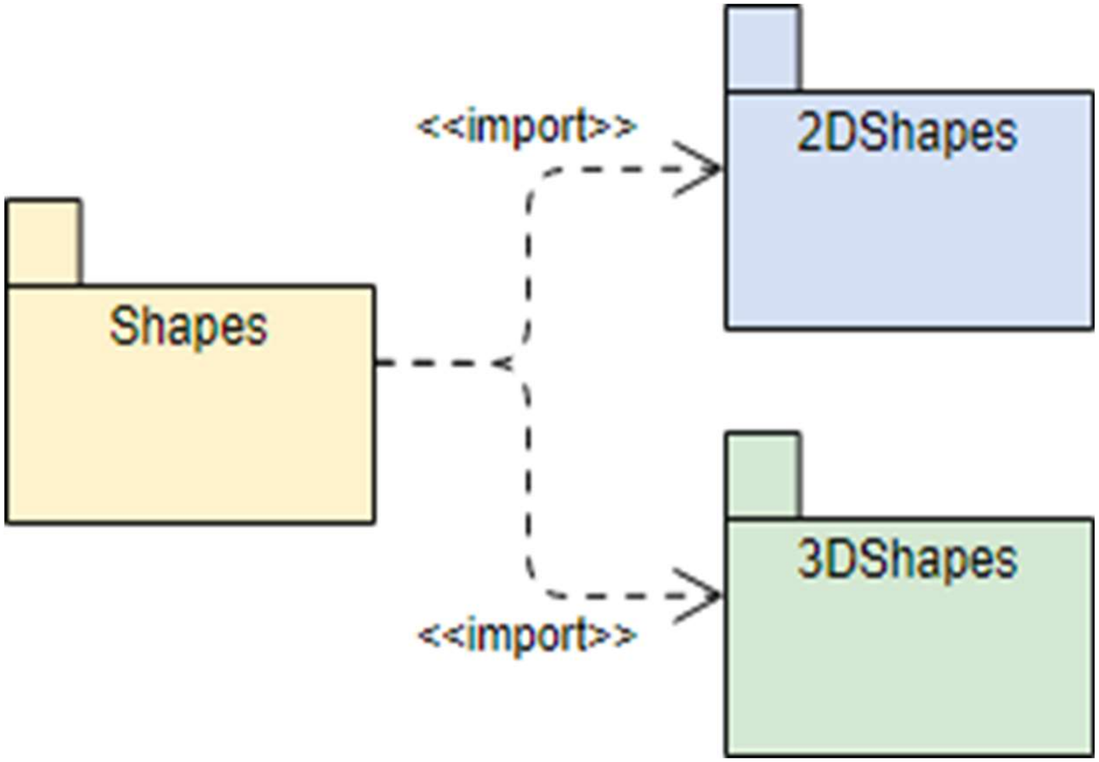
- Though there are two stereotypes users can use their own stereotype to represent the type of dependency between two packages.

- <<import>> - one package imports the functionality of other package



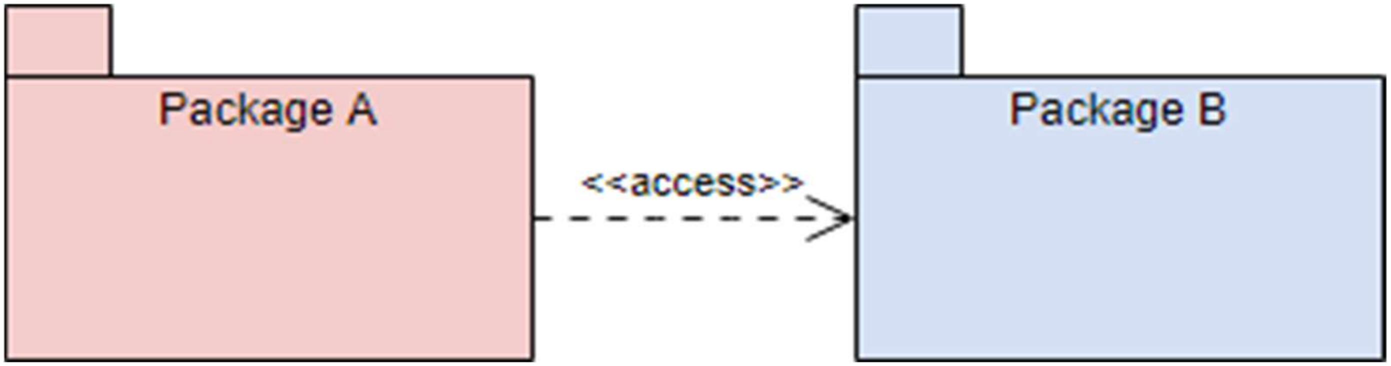
Dependency

- Example – <<import>> Dependency

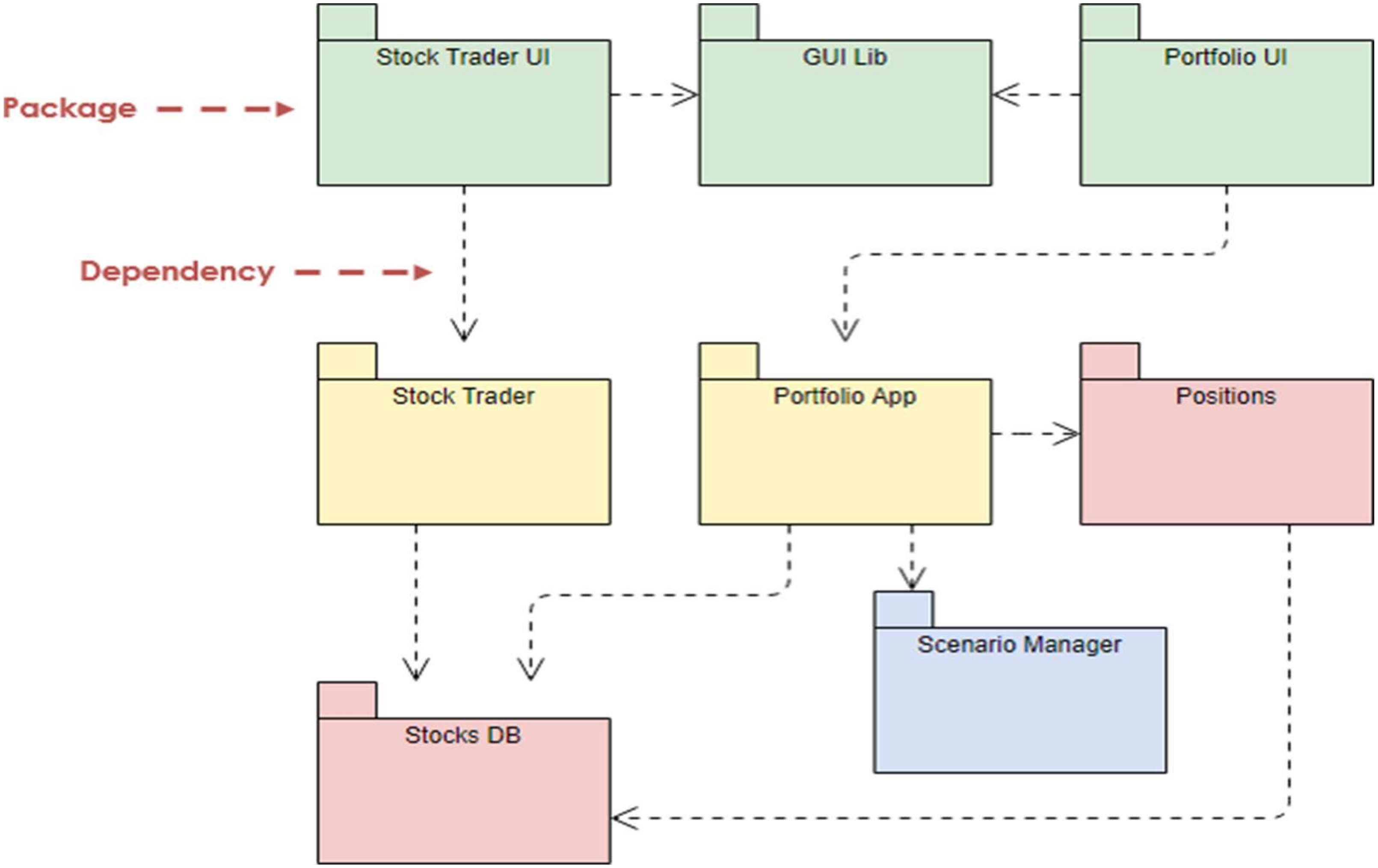

Shapes
2DShapes
3DShapes
<<import>>
<<import>>

Dependency

- <<access>> - one package requires help from functions of other package


Package A
<<access>>
Package B

Packages


Package
Dependency
Stock Trader UI
GUI Lib
Portfolio UI
Stock Trader
Portfolio App
Positions
Scenario Manager
Stocks DB

Packages

- package library;

- public class Program {

- public static void main(String[] args) {
- System.out.println("Hello packageworld!");
- }
- }

Packages

- Every package, including the default package, may contain other packages.
- For instance, in the package definition package library.domain the package domain is inside the package library.
- The word domain is often used to refer to the storage space of the classes that represent the concepts of the problem domain.
- For example, the class Book could be inside the package library.domain, since it represents a concept in the library application.

Packages

```
package library.domain;

public class Book {
    private String name;

    public Book(String name) {
        this.name = name;
    }

    public String getName() {
        return this.name;
    }
}
```

Packages

- A class can access classes inside a package by using the import statement.

- The class Book in the package library.domain is made available for use with the statement import library.domain.Book;.

- The import statements that are used to import classes are placed in the source code file after the package definition.

Packages

```
package library;
import library.domain.Book;
public class Program {
    public static void main(String[] args) {
        Book book = new Book("the ABCs of packages!");
        System.out.println("Hello packageworld: " + book.getName());
    }
```

```
Hello packageworld: the ABCs of packages!
```

Packages

- Every project you see in NetBeans is in your computer's file system or on a centralized server.
- The project directory src/main/java contains the source code files of the program.
- If the package of a class is library, that class is stored inside the src/main/java/libary folder of the source code directory.
- You can also check the concrete project structure in NetBeans in the **Files** tab, which is normally next to the **Project** tab.
- If you cannot see the **Files** tab, you can make it appear by choosing the option **Files** from the dropdown menu **Window**.
- Application development is normally done in the **Projects** tab, where NetBeans hides unimportant files from the programmer.

Packages and access modifiers

- Until now, we've used two access modifiers.
- The modifier private is used to define variables (and methods) that are only visible inside the class where they are defined.
- They cannot be used from outside that class.
- The methods and variables defined with public, on the other hand, are available for everyone to use.
- If you create an object of the UserInterface class above, its constructor and start method are callable from anywhere in the program.
- The method printTitle and the variable scanner are available only from within the class.

Packages and access modifiers

```
• package library.ui;

• public class UserInterface {
•     private Scanner scanner;

•     public UserInterface(Scanner scanner) {
•         this.scanner = scanner;
•     }

•     public void start() {
•         printTitle();

•         // other functionality
•     }

•     private void printTitle() {
•         System.out.println("***********");
•         System.out.println("* LIBRARY *");
•         System.out.println("***********");
•     }
• }
```

Packages and access modifiers

- If you create an object of the UserInterface class above, its constructor and start method are callable from anywhere in the program.

- The method printTitle and the variable scanner are available only from within the class.

- If the access modifier is missing, the methods and variables are only visible inside the same package.

- We call this the default or package modifier.

- Let's change the example above so that method printTitle has the package access modifier.

Packages and access modifiers

```
package library.ui;

public class UserInterface {
    private Scanner scanner;

    public UserInterface(Scanner scanner) {
        this.scanner = scanner;
    }

    public void start() {
        printTitle();

        // other functionality
    }

    void printTitle() {
        System.out.println("***********");
        System.out.println("* LIBRARY *");
        System.out.println("***********");
    }
}
```

Packages and access modifiers

- Now the classes inside the same package

- — i.e., the classes inside the package library.ui — can use the method printTitle.

```
• package library.ui;

• import java.util.Scanner;

• public class Main {

•     public static void main(String[] args) {
•         Scanner scanner = new Scanner(System.in);
•         UserInterface ui = new UserInterface(scanner);

•         ui.printTitle(); // works!
•     }
• }
```

Packages and access modifiers

- If a class is in a different package, the method printTitle cannot be called.
- In the example below, the class Main is in the package library.
- As the printTitle method is in the package library.ui and has the package access modifier, it cannot be used.

```
package library;

import java.util.Scanner;
import library.ui.UserInterface;

public class Main {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        UserInterface ui = new UserInterface(scanner);

        ui.printTitle(); // doesn't work!
    }
}
```

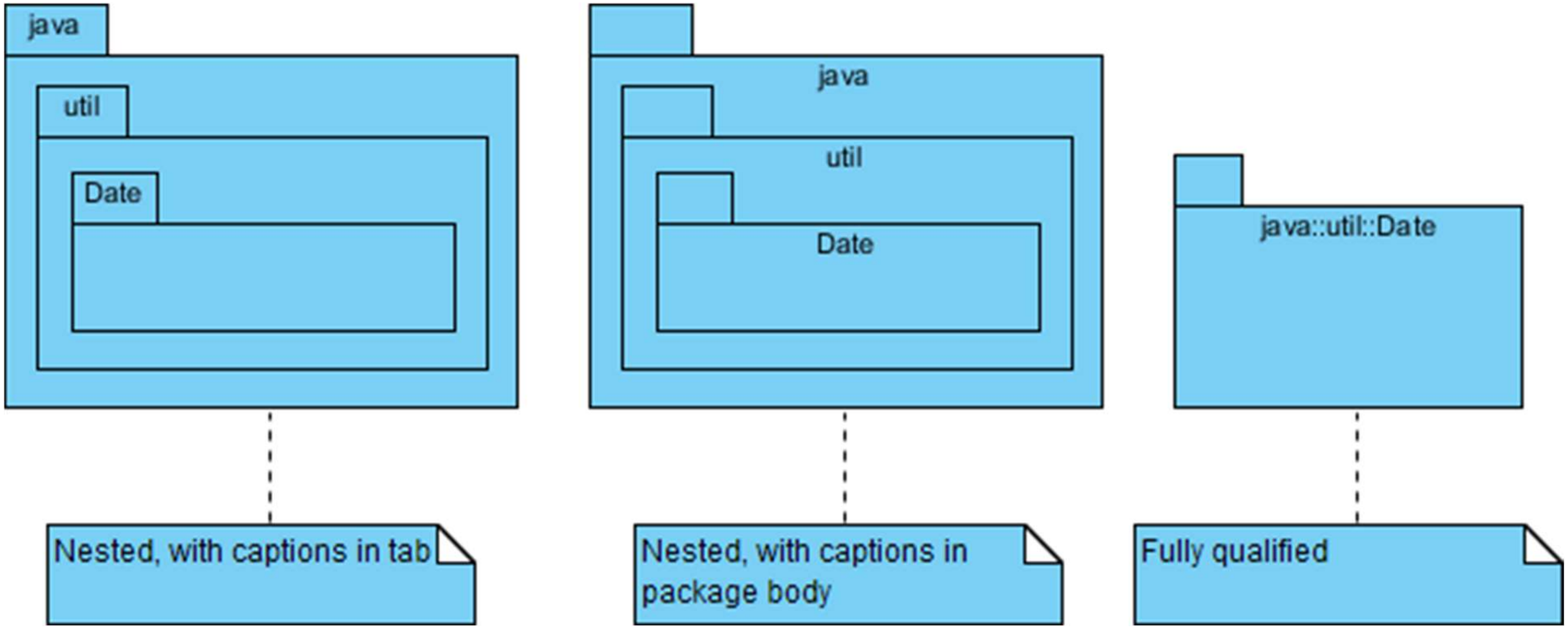
Basic Concepts of Package Diagram

- Package diagram follows hierarchal structure of nested packages.
- Atomic module for nested package are usually class diagrams.
- There are few constraints while using package diagrams, they are as follows.
- Package name should not be the same for a system, however classes inside different packages could have the same name.
- Packages can include whole diagrams, name of components alone or no components at all.
- Fully qualified name of a package has the following syntax.

Name owing the package : : Name of the package

java : : util : : Date

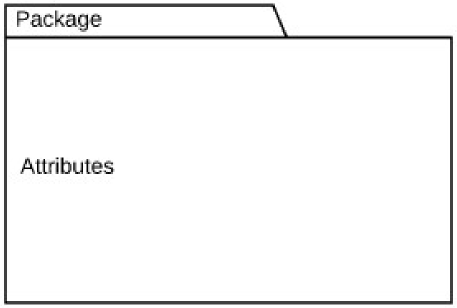
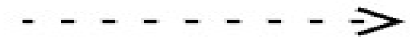
Packages can be represented by the notations with some examples shown below:


java
util
Date
java
util
Date
java::util::Date
Nested, with captions in tab
Nested, with captions in package body
Fully qualified

Basic components of a package diagram

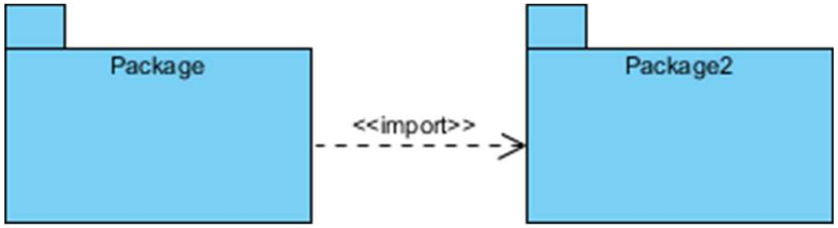
- The makeup of a package diagram is relatively simple.

- Each diagram includes only two symbols:

	Package	Groups common elements based on data, behavior, or user interaction
	Dependency	Depicts the relationship between one element (package, named element, etc.,) and another

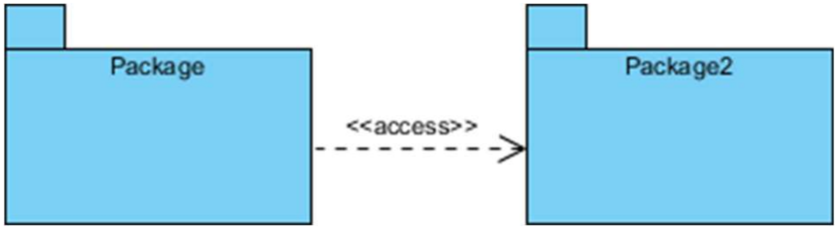
- Package Diagram Example – Import

- <<import>> - one package imports the functionality of other package


Package
<<import>>
Package2

Package Diagram Example – Access

<<access>> - one package requires help from functions of other package.


Package
<<access>>
Package2

- **Package**: A namespace used to group together logically related elements within a system.

- Each element contained within the package should be a packageable element and have a unique name.

- **Packageable element**: A named element, possibly owned directly by a package.

- These can include events, components, use cases, and packages themselves.

- Packageable elements can also be rendered as a rectangle within a package, labeled with the appropriate name.

- **Dependencies**: A visual representation of how one element (or set of elements) depends on or influences another.

- Dependencies are divided into two groups: access and import dependencies.

- **Element import**: A directed relationship between an importing namespace and an imported packageable element.

- This is used to import select individual elements without resorting to a package import and without making it public within the namespace.

- **Package import**: A directed relationship between and importing namespace and an imported package.

- This type of directed relationship adds the names of the members of the imported package to its own namespace

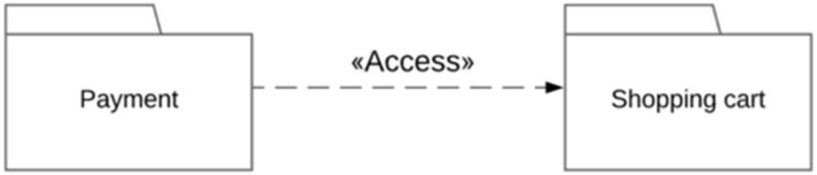
- **Package merge**: A directed relationship in which the contents of one package are extended by the contents of another.

- Essentially, the content of two packages are combined to produce a new package.

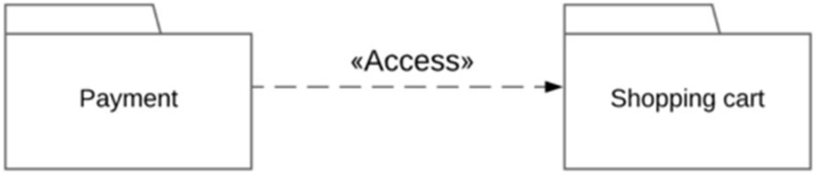
Dependency notations in a package diagram

- There are two main types of dependencies:

- **Access**: Indicates that one package requires assistance from the functions of another package. Example:


«Access»
Payment
Shopping cart

Import: Indicates that functionality has been imported from one package to another.
Example:


«Access»
Payment
Shopping cart

- Dependencies can also be broken down further into the following categories:

- **Usage**: Occurs when a given named element requires another for its full definition and deployment.

- Example: client and supplier.

- **Abstraction**: Relates two elements representing the same concept at different levels of abstraction within the system (usually a relationship between client and supplier).

- **Deployment**: Depicts the deployment of an artifact to a deployment target.

When to draw Package Diagram?

- The UML does not treat package diagrams as a separate technique, It is often useful to combine them by grouping other model elements together into different packages on the same diagram.

- Package diagrams can be useful in many ways, such as:

- To create an overview of a large set of model elements

- To organize a large model

- To group related elements

- To separate namespaces

Criteria for Decomposing a System into Packages

- Different owners - who is responsible for working on which diagrams?

- Different applications - each problem has its own obvious partitions;

- Clusters of classes with strong cohesion - e.g., course, course description, instructor, student,...

- Or: use an architectural pattern to help find a suitable decomposition such as MVC Framework

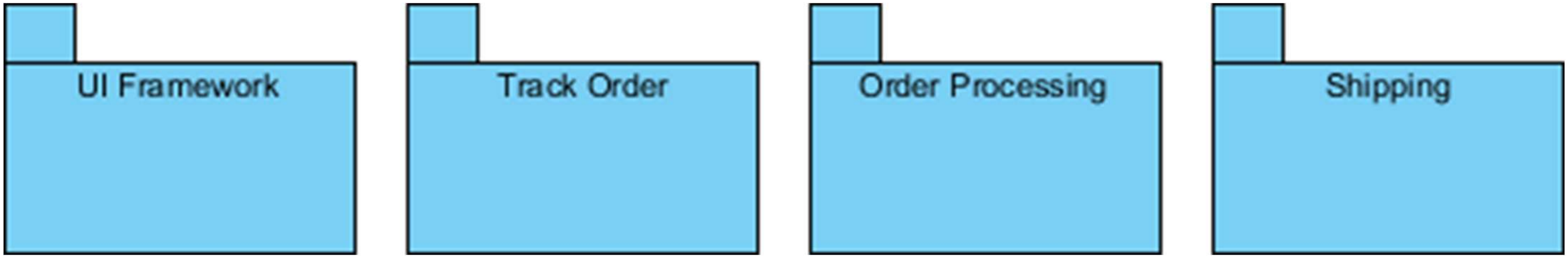
Other Guidelines for Packages

- Gather model elements with strong cohesion in one package
- Keep model elements with low coupling in different packages
- Minimize relationships, especially associations, between model elements in different packages
- Namespace implication: an element imported into a package does not "know" how it is used in the imported package

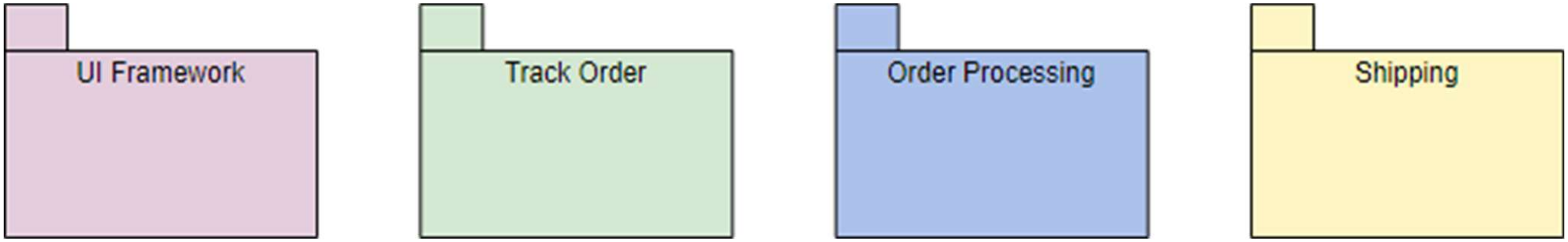
Order Processing System - The Problem Description

- We are going to design package diagram for "Track Order" scenario for an online shopping store.
- Track Order module is responsible for providing tracking information for the products ordered by customers.
- Customer types in the tracking serial number, Track Order modules refers the system and updates the current shipping status to the customer.
- Based on the project description we should first identify the packages in the system and then related them together according to the relationship:
- Identify the packages of the system
- There is a track order module, it has to talk with other module to know about the order details, let us call it "Order Details".
- Next after fetching Order Details it has to know about shipping details, let us call that as "Shipping".

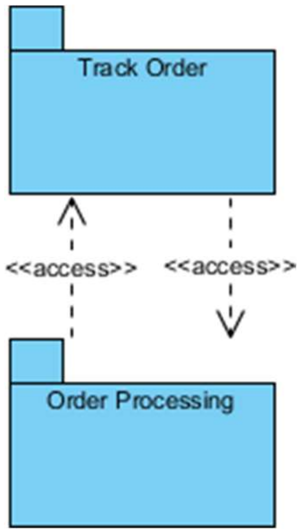
Package Diagram Example - Order Processing System

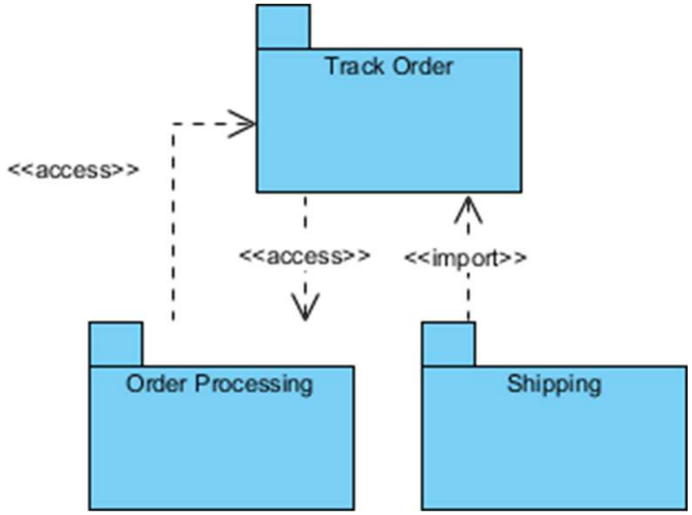

UI Framework
Track Order
Order Processing
Shipping

Package Diagram Example - Order Processing System


UI Framework
Track Order
Order Processing
Shipping

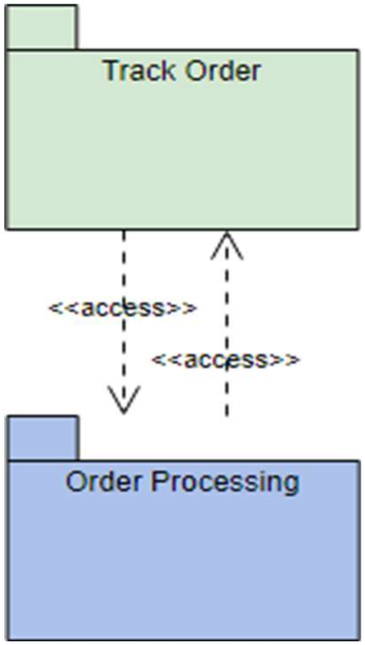
- identify the dependencies in the System
- Track order should get order details from "Order Details" and "Order Details" has to know the tracking info given by the customer.
- Two modules are accessing each other which suffices <<access>> dual dependency

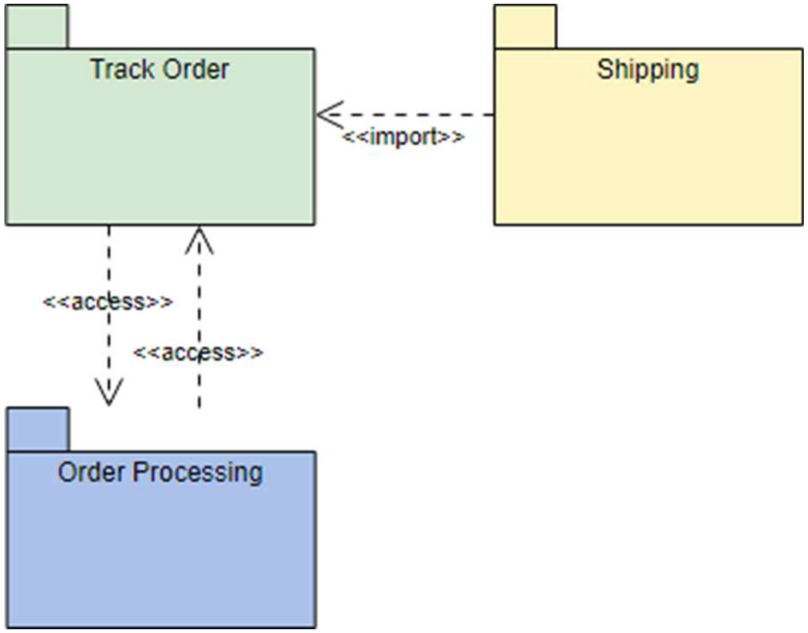

Track Order
<<access>>
<<access>>
Order Processing


Track Order
<<access>>
<<access>>
<<import>>
Order Processing
Shipping

To know shipping information, "Shipping" can import "Track Order" to make the navigation easier.

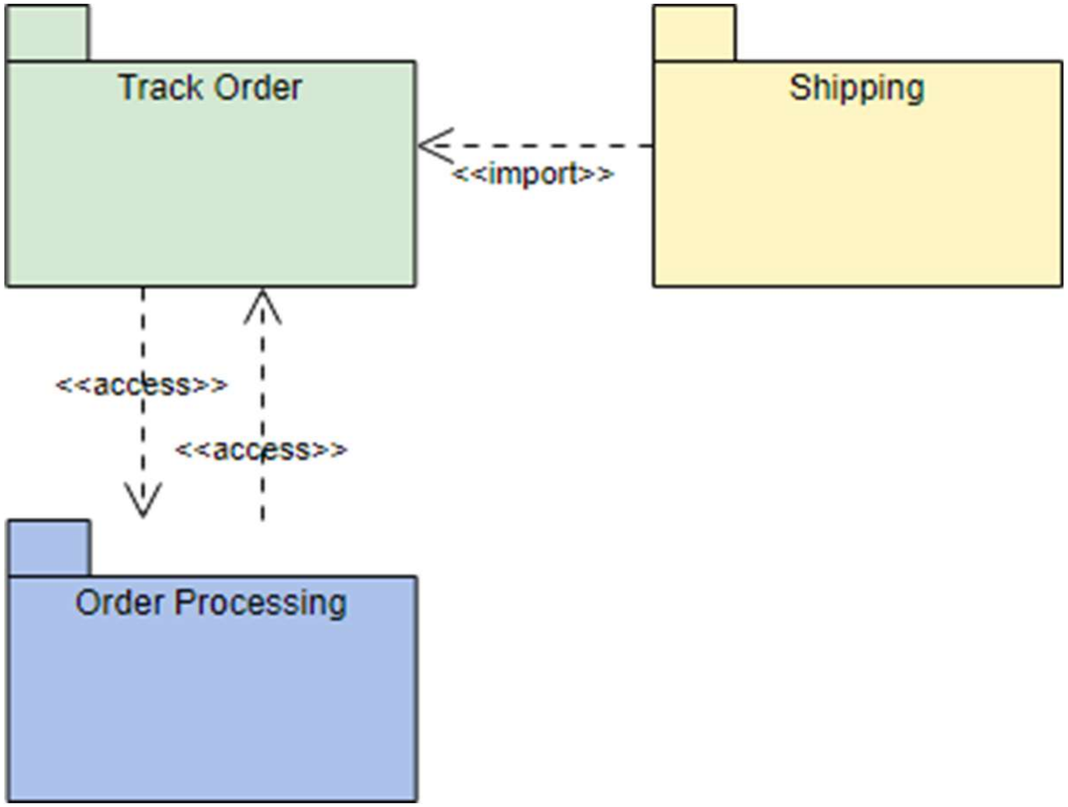
- identify the dependencies in the System
- Track order should get order details from "Order Details" and "Order Details" has to know the tracking info given by the customer.
- Two modules are accessing each other which suffices <<access>> dual dependency


Track Order
<<access>>
<<access>>
Order Processing

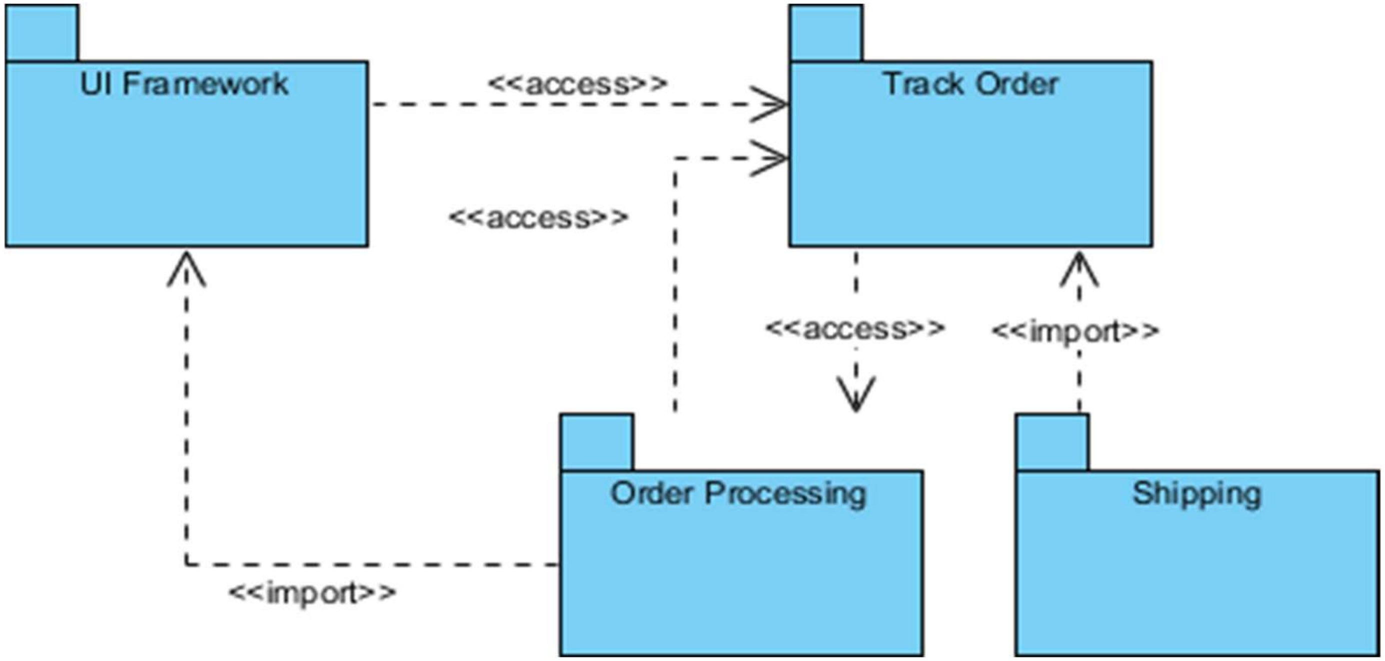

Track Order
Shipping
<<import>>
<<access>>
<<access>>
Order Processing

To know shipping information, "Shipping" can import "Track Order" to make the navigation easier.

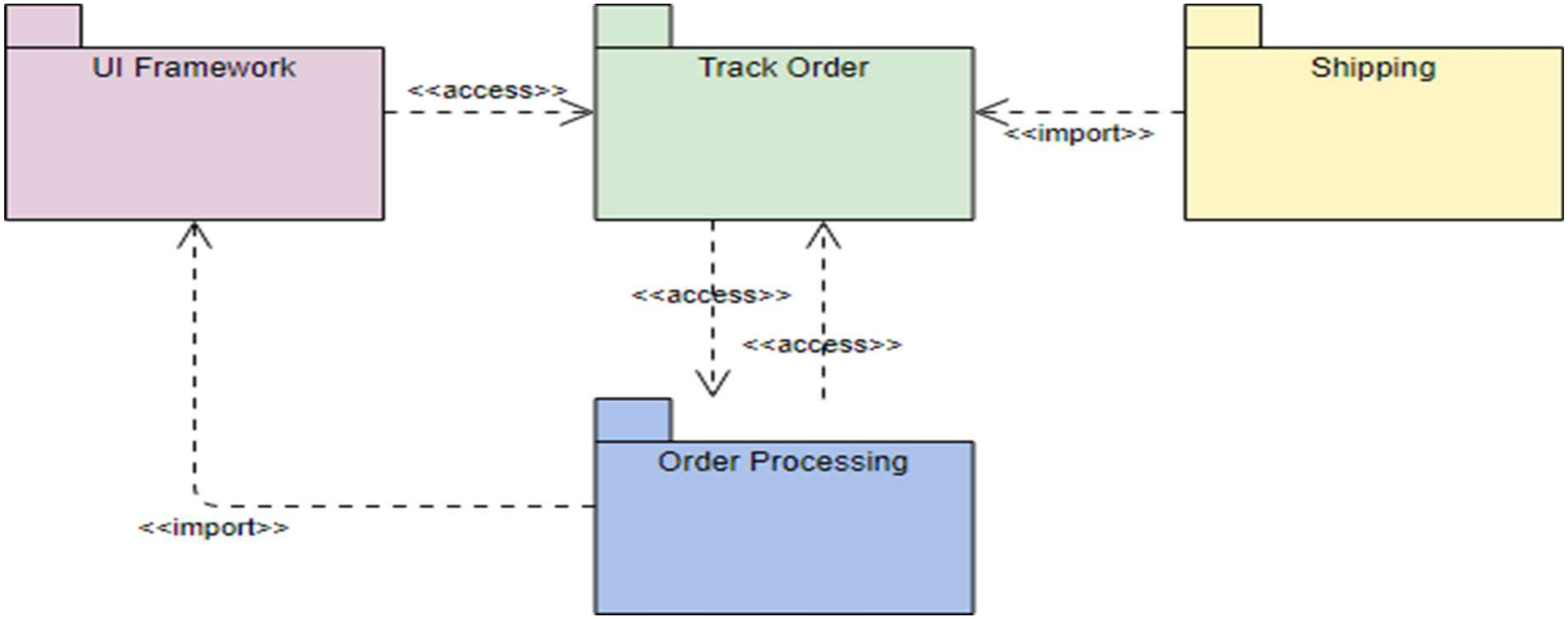
- To know shipping information, "Shipping" requires to import "Track Order" to complete the shipping process.


Track Order
Shipping
<<import>>
<<access>>
<<access>>
Order Processing

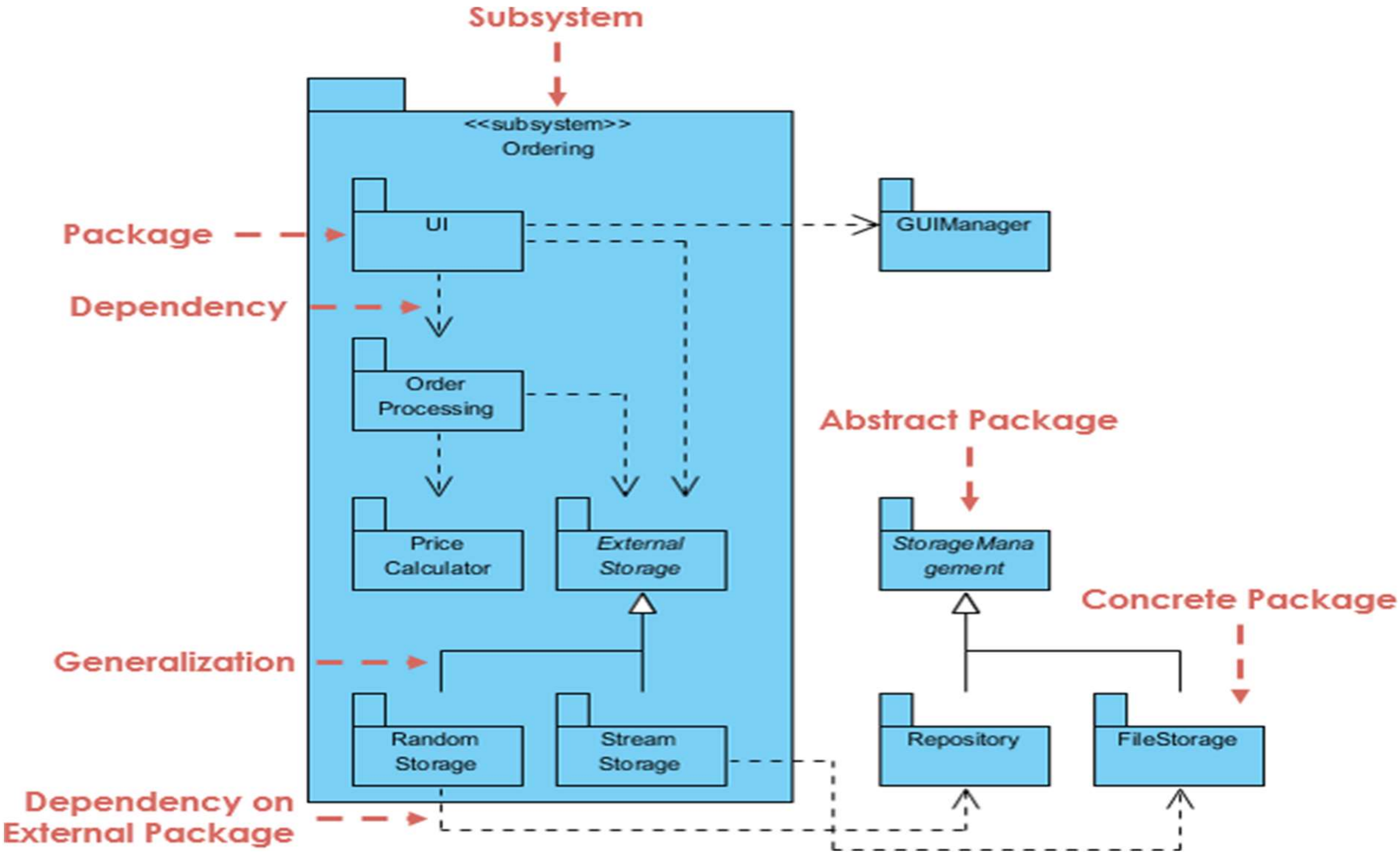
Track Order dependency to UI Framework is also mapped which completes our Package Diagram for Order Processing subsystem.


UI Framework
<<access>>
Track Order
<<access>>
<<access>>
<<import>>
Order Processing
Shipping
<<import>>

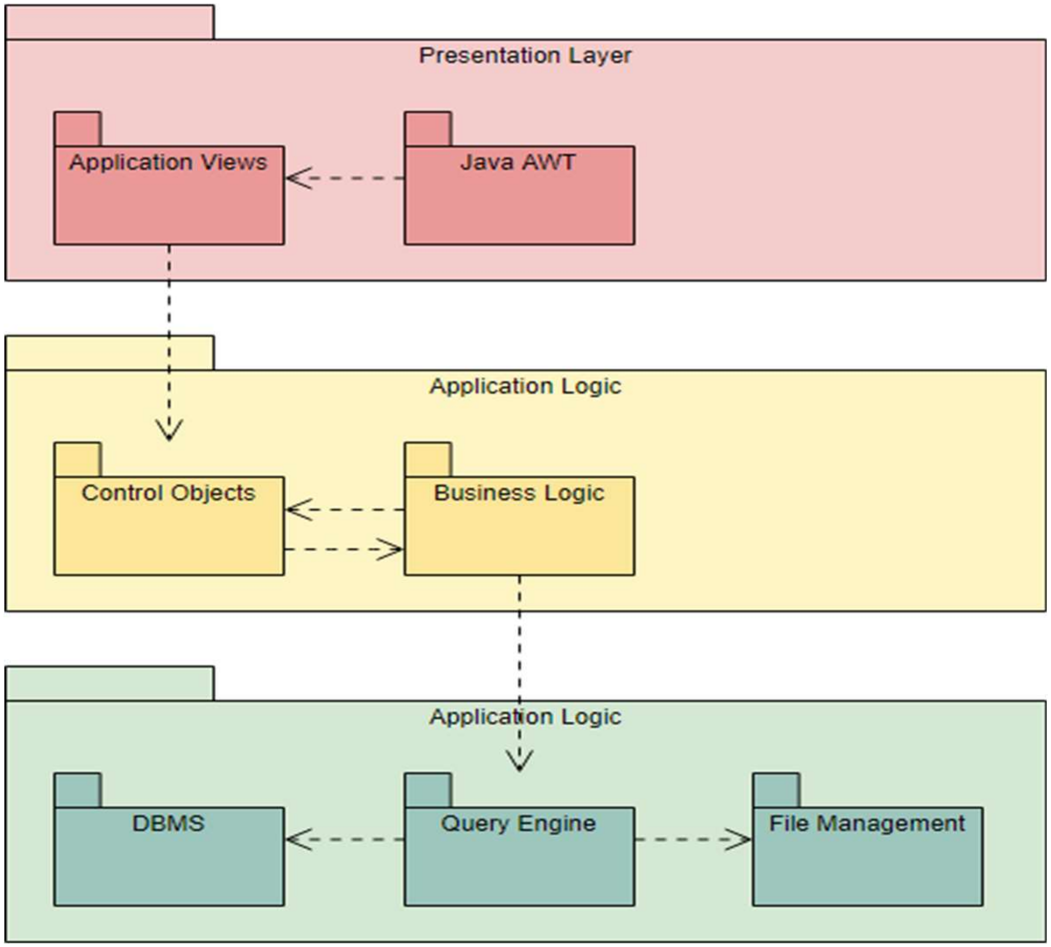
Track Order dependency to UI Framework is also mapped which completes our Package Diagram for Order Processing subsystem.


UI Framework
Track Order
Shipping
Order Processing
<<access>>
<<import>>
<<access>>
<<access>>
<<import>>

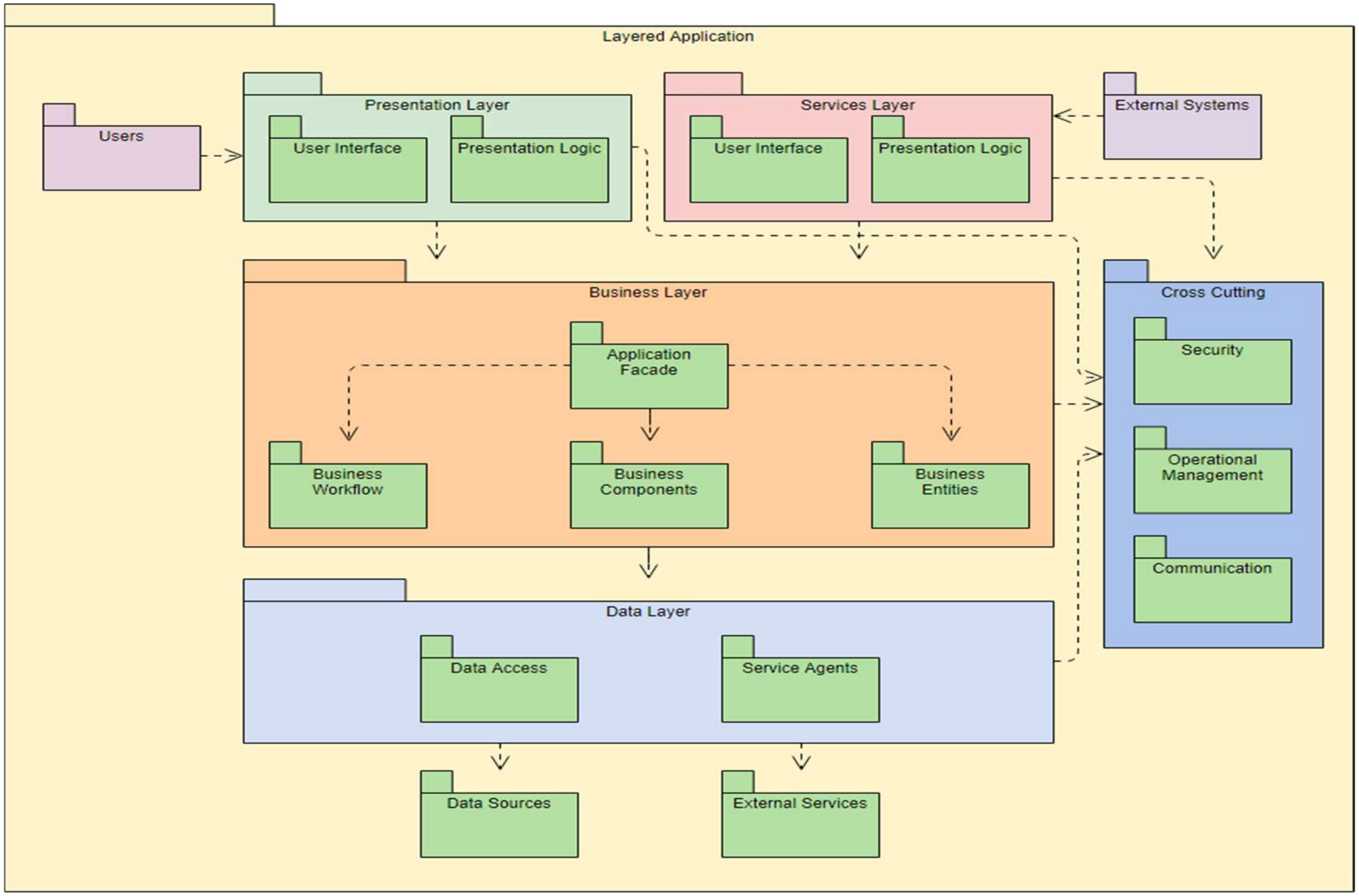
Package Diagram Example - Order Subsystem


Subsystem
<<subsystem>>
Ordering
Package
UI
GUIManager
Dependency
Order Processing
Abstract Package
Price Calculator
External Storage
StorageManagement
Concrete Package
Generalization
Random Storage
Stream Storage
Repository
FileStorage
Dependency on External Package

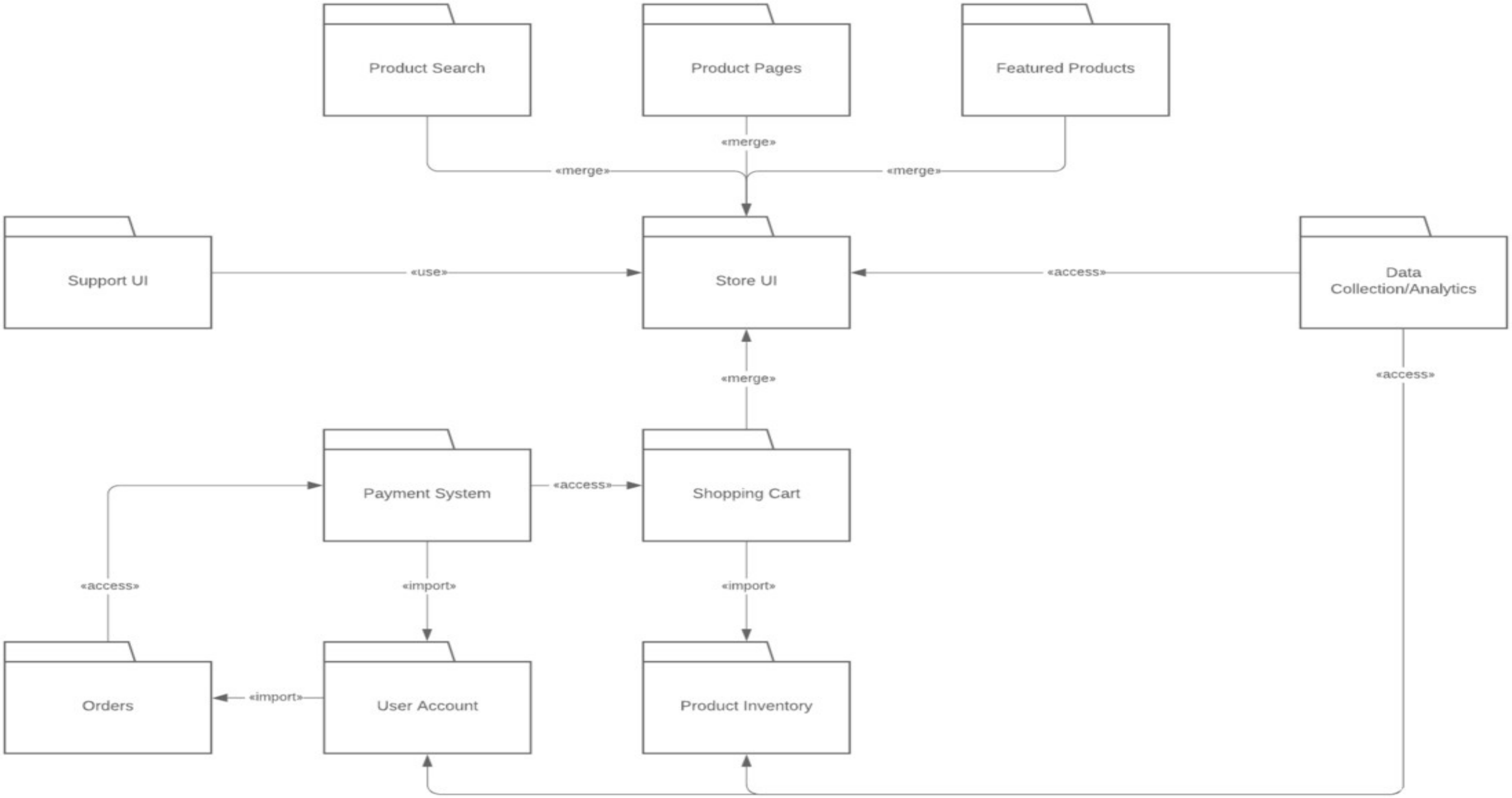
Package Diagram Example – MVC Structure


Presentation Layer
Application Views
Java AWT
Application Logic
Control Objects
Business Logic
Application Logic
DBMS
Query Engine
File Management

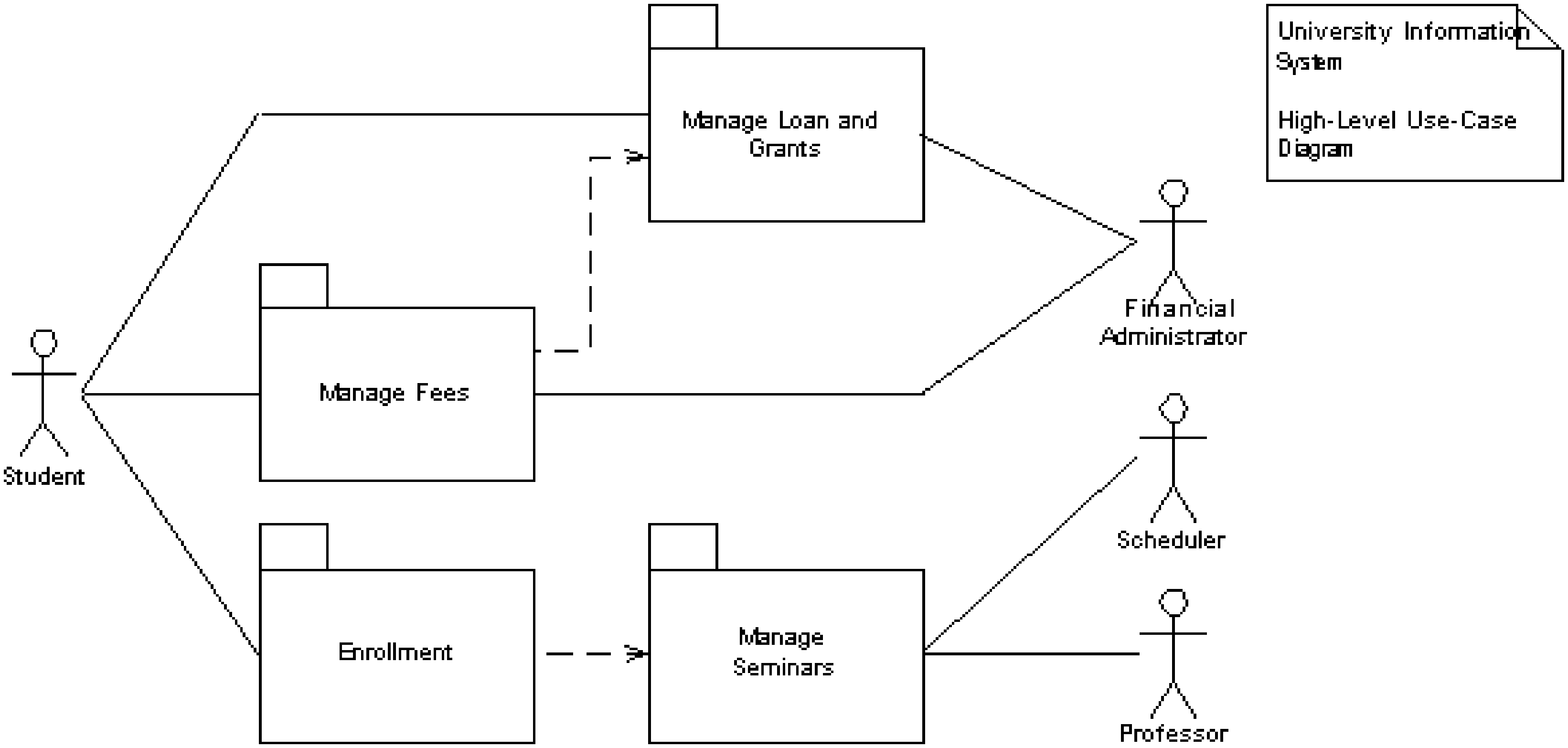
Package Diagram Example – Layering Structure


Layered Application
Users
Presentation Layer
User Interface
Presentation Logic
Services Layer
User Interface
Presentation Logic
External Systems
Business Layer
Application Facade
Business Workflow
Business Components
Business Entities
Cross Cutting
Security
Operational Management
Communication
Data Layer
Data Access
Service Agents
Data Sources
External Services

Shopping Cart


Product Search
Product Pages
Featured Products
«merge»
«merge»
«merge»
Support UI
«use»
Store UI
«access»
Data Collection/Analytics
«merge»
«access»
Payment System
«access»
Shopping Cart
«access»
«import»
«import»
Orders
«import»
User Account
Product Inventory

Use Case Package Example


Manage Loan and Grants
Manage Fees
Enrollment
Manage Seminars
Student
Financial Administrator
Scheduler
Professor
University Information System

High-Level Use-Case Diagram

General Guidelines

- Give Packages Simple, Descriptive Names
- Apply Packages to Simplify Diagrams
- Packages Should be Cohesive
- Indicate Architectural Layers With Stereotypes on Packages
- Avoid Cyclic Dependencies Between Packages
- Package Dependencies Should Reflect Internal Relationships

Benefits of a package diagram

- A well-designed package diagram provides numerous benefits to those looking to create a visualization of their UML system or project.
- They provide a clear view of the hierarchical structure of the various UML elements within a given system.
- These diagrams can simplify complex class diagrams into well-ordered visuals.
- They offer valuable high-level visibility into large-scale projects and systems.
- Package diagrams can be used to visually clarify a wide variety of projects and systems.
- These visuals can be easily updated as systems and projects evolve.

BITS Pilani
Pilani Campus

SOFTWARE DESIGN PRINCIPLES
Realizing Use Cases

By Prof A R Rahman

Realizing Use Cases

What is a Use Case Realization?

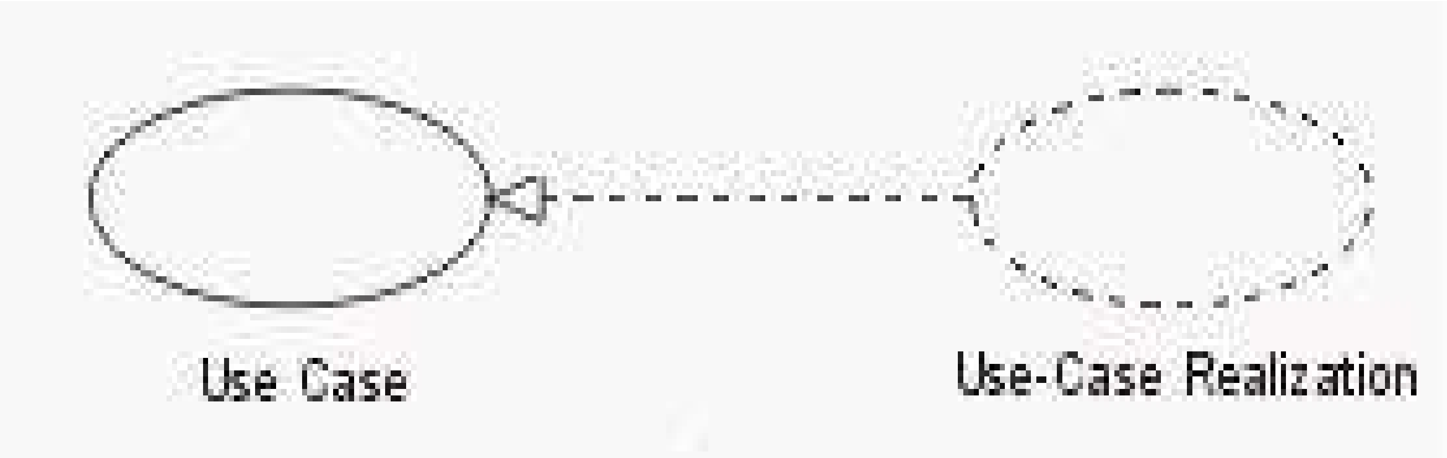
- A use case realization provides a construct to organize artifacts which shown how the physical design of a system supports the logical business behavior outlined by a use case.

- To show this traceability between the logical and physical design, in the use case diagram each use case is depicted as an oval shown with a solid-line border.

- Then for each use case can you may show a use case realization as an oval with a dotted-line border.

- The use case and its corresponding use case realization are linked using a realization dependency which is shown in UML as a dashed line with a triangular arrowhead at the end corresponding to the realized element (the use case).

What is a Use Case Realization?


Use Case
Use-Case Realization

What is a Use Case Realization?

- Each use case realization will define the physical design in terms of classes and collaborating objects which support the use case.

- Therefore, each use case realization typically is made up of a class diagram and a number of interaction diagrams, most commonly sequence diagrams, showing the collaboration or interaction between physical objects.

- Use case realizations allow the analyst to clearly separate the concerns of the logical and physical design.

- Since a logical design (the business behavior and requirements) can be implemented or realized via a number of different physical implementations, if a physical design changes the logical use case can remain unaffected.

- Also, a use case realization is an excellent form of requirements traceability from the logical business requirements down to the physical implementation of the solution.

What is a Use Case Realization?

- According to RUP the Use Case Realization task creates the Analysis Model consisting of Boundary classes, Control classes and Entity classes diagrammed in Class Diagrams, and in Sequence Diagrams.

- These Models are LOGICAL models describing white box inner workings of the system (as opposed to the use cases that treat the system as a Black box.

- The Design Model - that is often created by adding details and modifying the Analysis Model - represents the PHYSICAL description of the system to be done.

- One major difference between the Analysis Model and the Design Model, is that the Design Model contain implementation language specific information, whereas the Analysis Model does not.

- Use Cases are the WHAT, the functional requirements. Use Case Realizations are the HOW, the solution (at the logical level)

use-case realization

- We have identified and modeled the requirements to the system in the previous sessions

- How the results from the analysis should be transferred into the design in a structured and reproducible way

- That's the only way to ensure that all results from the analysis will be taken into account, and that changes to the requirements will be made in the right places, or changes to the design will be harmonized with the requirements.

use-case realization

- We are only looking at the abstract solution, i.e., structure and behavior descriptions, excluding concrete implementations.

- This means that we do not take the technical aspects, such as concrete hardware, software technologies, and so on, into account.

- We will add these technical aspects later on when we'll be ready to take technological decisions on the basis of the abstract solution.

use-case realization

- What is meant by use case realization

- Two approaches for realizing use cases:

– Robustness analysis combined with communication diagrams

– Class-Responsibility-Collaboration (CRC)

- How to combine use case class diagrams into a single analysis class model

use-case realization

- A use-case realization represents how a use case will be implemented in terms of collaborating objects.
- The realizations reside within the design.
- By walking through a design exercise of showing how the design elements will perform the use case, the team gets confirmation that the design is robust enough to perform the required behavior.
- The realization can take various forms.
- It may include, for example, a textual description (a document), class diagrams of participating classes and subsystems, and interaction diagrams (communication and sequence diagrams) that illustrate the flow of interactions between class and subsystem instances.

use-case realization

- The reason for separating the use-case realization from its use case is that doing so allows the requirements, in the form of use cases, to be managed separately from the design, in the form of realizations.

- This decoupling will prove invaluable if the architecture is changed enough that the realization needs to be reworked while the requirements remain unaffected.

- Even without such a circumstance, the clear separation of concerns between requirements and design is valuable.

- In a model, a use-case realization is represented as a UML (Unified Modeling Language) collaboration that groups the diagrams and other information (such as textual descriptions) that form the use-case realization.

use-case realization

- Like other aspects of the design, the UML diagrams that support use-case realizations can be produced at various levels of abstraction.

- A first pass in creating the realization might produce a diagram at an analysis level of abstraction where the participants will be high-level elements that are expected to be revisited and detailed down to the design level in a second pass.

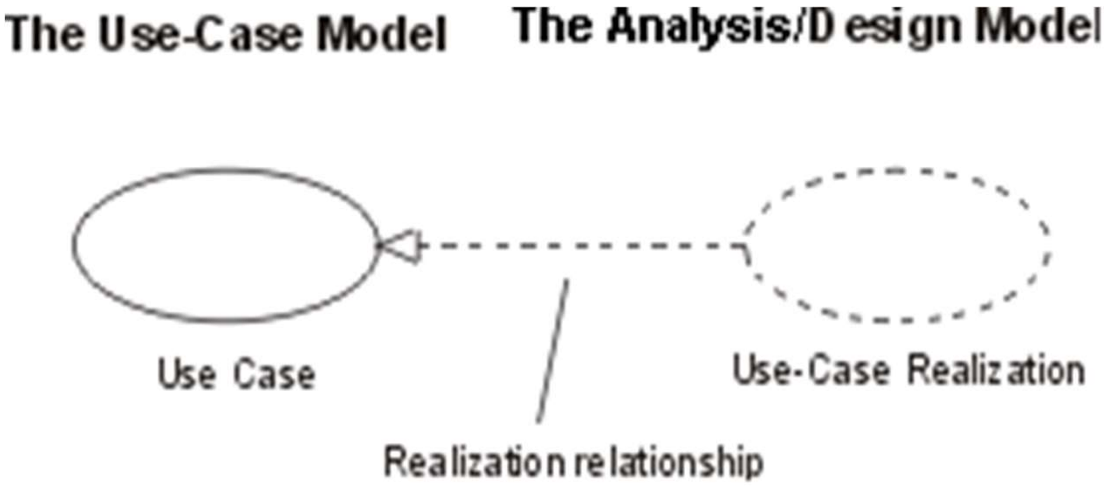
- If the architecture and design idioms are well-understood, the realization could immediately be created at a low level of abstraction that specifies more detail on the elements and how they will collaborate to realize the behavior of the use case.

- In the latter case, it is valuable to model patterns and architectural mechanisms to reduce the amount of low-level detail in each realization.

The UML notation for use-case realization

- For each use case in the requirements, there can be a use-case realization in the design with a realization relationship to the use case, as the following figure shows.

- In UML, this is shown as a dashed arrow with an arrowhead, like a generalization relationship, indicating that a realization is a kind of inheritance, as well as a dependency.


The Use-Case Model
The Analysis/Design Model
Use Case
Use-Case Realization
Realization relationship

Class diagrams owned by a use case realization

- For each use-case realization, there may be one or more class diagrams that depict its participating classes.

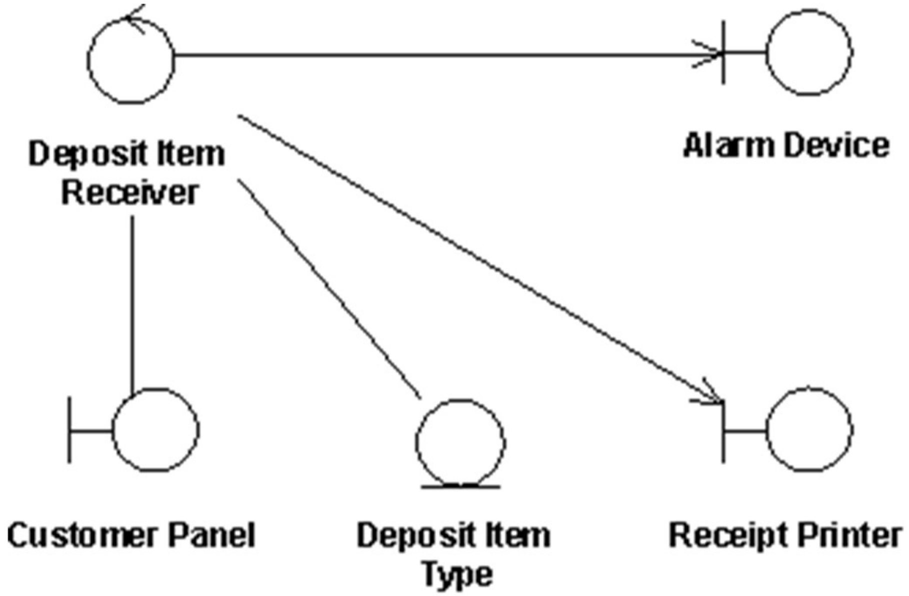
- A class and its objects often participate in several use-case realizations.

- While designing, it is important to coordinate all of the requirements related to a class that different use-case realizations may have.

- The figure below shows an analysis class diagram for the realization of the Receive Deposit Item use case.

- Notice the use of boundary-control-entity stereotypes to represent analysis classes

The use case Receive Deposit Item and its analysis-level class diagram


Deposit Item Receiver
Alarm Device
Customer Panel
Deposit Item Type
Receipt Printer

Sequence and communication diagrams

- For each use-case realization, there can be one or more interaction diagrams that depict the participating objects and their interactions.
- There are two types of interaction diagrams: *sequence* diagrams and *communication* diagrams.
- They express similar information, but show it in different ways.
- Sequence diagrams show the explicit sequence of messages and are better when it is important to visualize the time order of messages.
- Communication diagrams show the communication links between objects and are better for understanding all of the effects on a given object and for algorithm design.

use-case realization

- Realizing use cases through interaction diagrams helps to keep the design simple and cohesive.

- Assigning responsibilities to classes on the basis of what the use-case scenario explicitly requires encourages the design to contain the following elements:

- Only the functionality actually used in support of a use-case scenario

use-case realization

- Functionality that can be tested through an associated test case

- Functionality that is more easily traceable to requirements and changes

- Explicitly declared class dependencies that are easier to manage

- These factors help improve the overall quality of the system.

From Requirements to Classes

- Requirements (use cases) are usually expressed in user language

- Use cases are units of development, but they are not structured like software

- The software we will implement consists of classes

- We need a way to translate requirements into classes

Goal of Realization

- An analysis class diagram is only an interim product
- This in turn will be realized as a design class diagram
- The ultimate product of realization is the software implementation of that use case

Communication Diagram Approach

- Analyse one use case at a time

- Identify likely classes involved (the use case collaboration)

– These may come from a domain model

- Draw a communication diagram that fulfils the needs of the use case

- Translate this into a use case class diagram

- Repeat for other use cases

Robustness Analysis

- Aims to produce set of classes robust enough to meet requirements of a use case

- Makes some assumptions about the interaction:

– Assumes some class or classes are needed to handle the user interface

– Abstracts logic of the use case away from entity classes (that store persistent data)

Robustness Analysis: Class Stereotypes

Class stereotypes differentiate the roles objects can play:

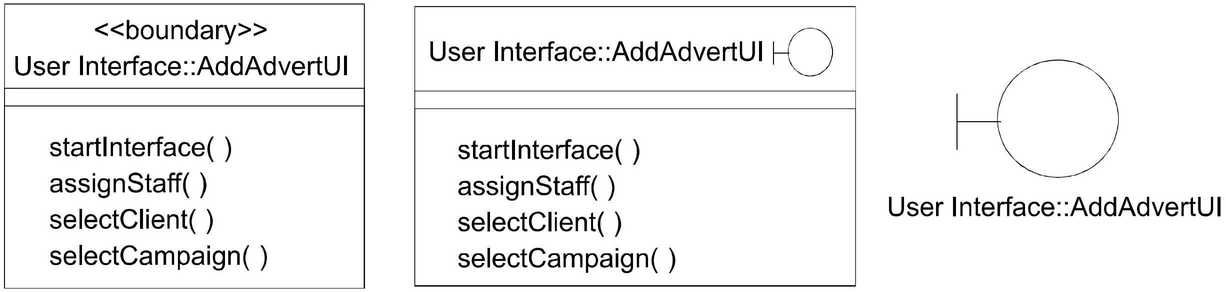
- Boundary objects model interaction between the system and actors (and other systems)
- Control objects co-ordinate and control other objects
- Entity objects represent information and behaviour in the application domain
- Entity classes may be imported from domain model
- Boundary and control classes are more likely to be unique to one application

Boundary Class Stereotype

- Boundary classes represent interaction with the user

- likely to be unique to the use case but inherited from a library

- Alternative notations:

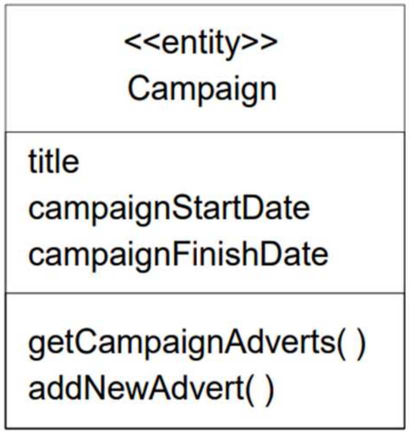


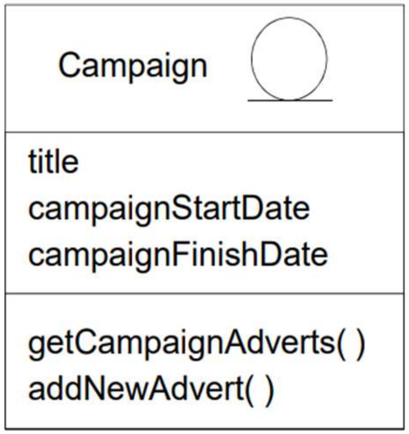
<<boundary>>
User Interface::AddAdvertUI
startInterface( )
assignStaff( )
selectClient( )
selectCampaign( )
User Interface::AddAdvertUI
startInterface( )
assignStaff( )
selectClient( )
selectCampaign( )
User Interface::AddAdvertUI

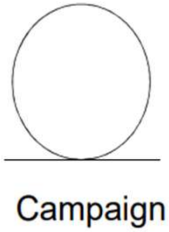
Entity Class Stereotype

- Entity classes represent persistent data and common behaviour likely to be used in more than one application system

- Alternative notations :

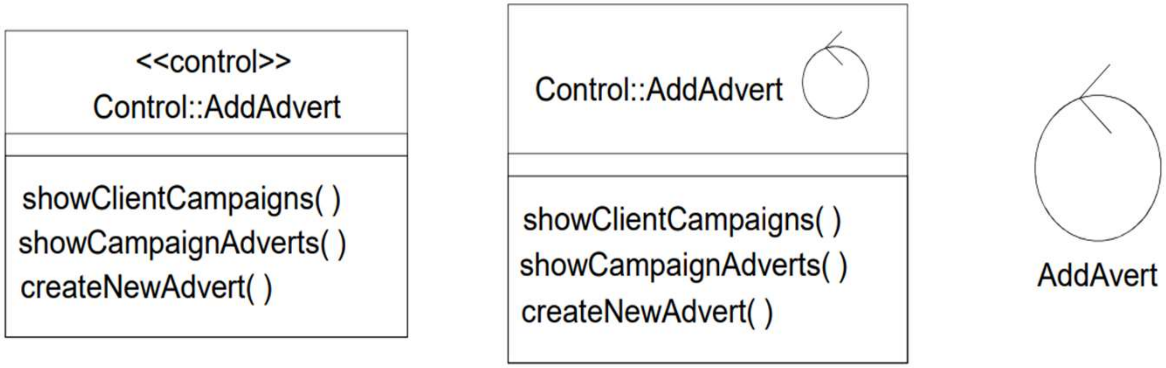

<<entity>>
Campaign
title
campaignStartDate
campaignFinishDate
getCampaignAdverts( )
addNewAdvert( )


Campaign
title
campaignStartDate
campaignFinishDate
getCampaignAdverts( )
addNewAdvert( )

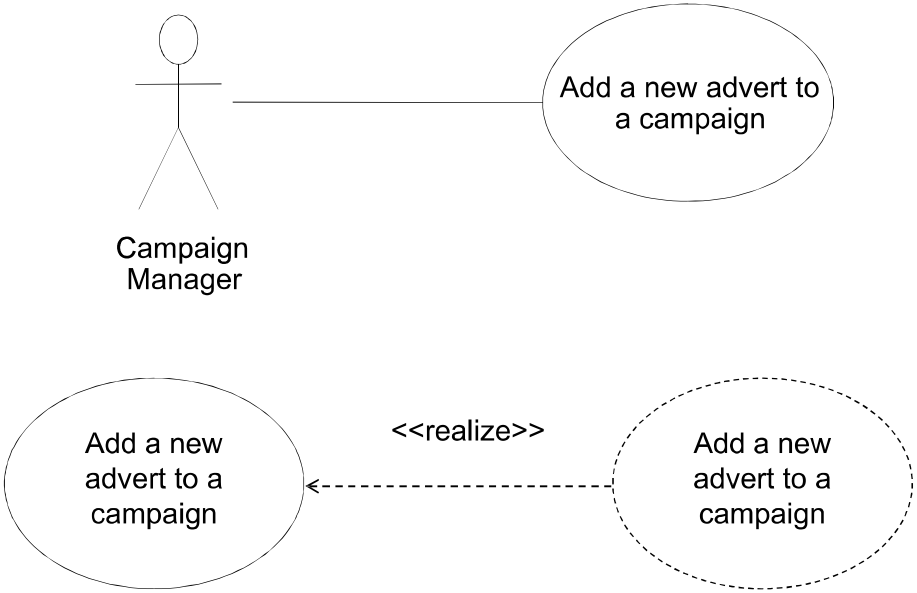


Control Class Stereotype

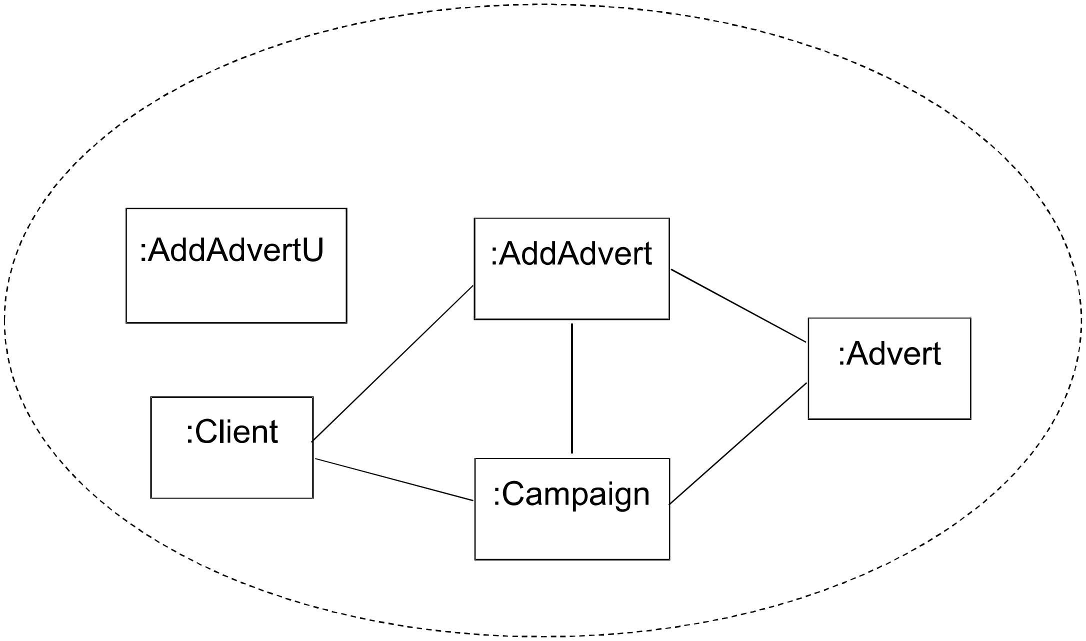
- Control classes encapsulate unique behaviour of a use case
- Specific logic kept separate from the common behaviour of entity classes
- Alternative notations:


<<control>>
Control::AddAdvert
showClientCampaigns( )
showCampaignAdverts( )
createNewAdvert( )
Control::AddAdvert
showClientCampaigns( )
showCampaignAdverts( )
createNewAdvert( )
AddAvert

Use Case and Collaboration


Add a new advert to a campaign
Campaign Manager
Add a new advert to a campaign
<<realize>>
Add a new advert to a campaign

A Possible collaboration



:AddAdvertU

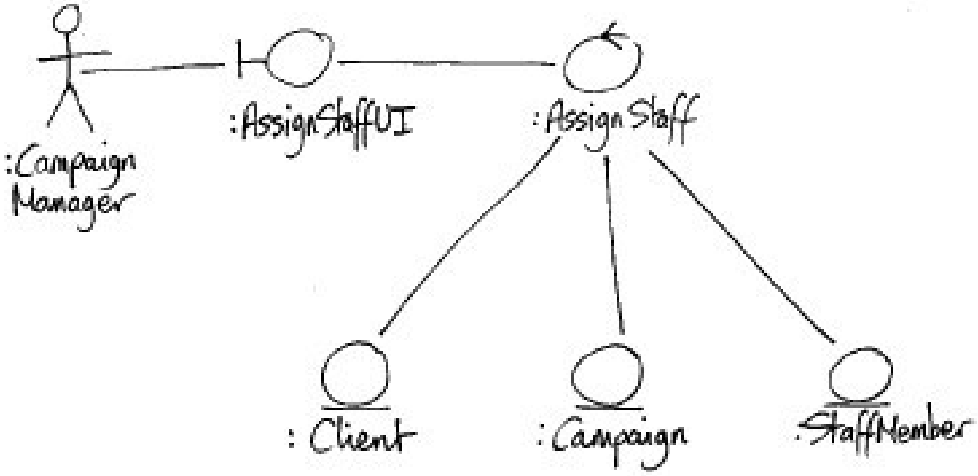
:AddAdvert

:Advert

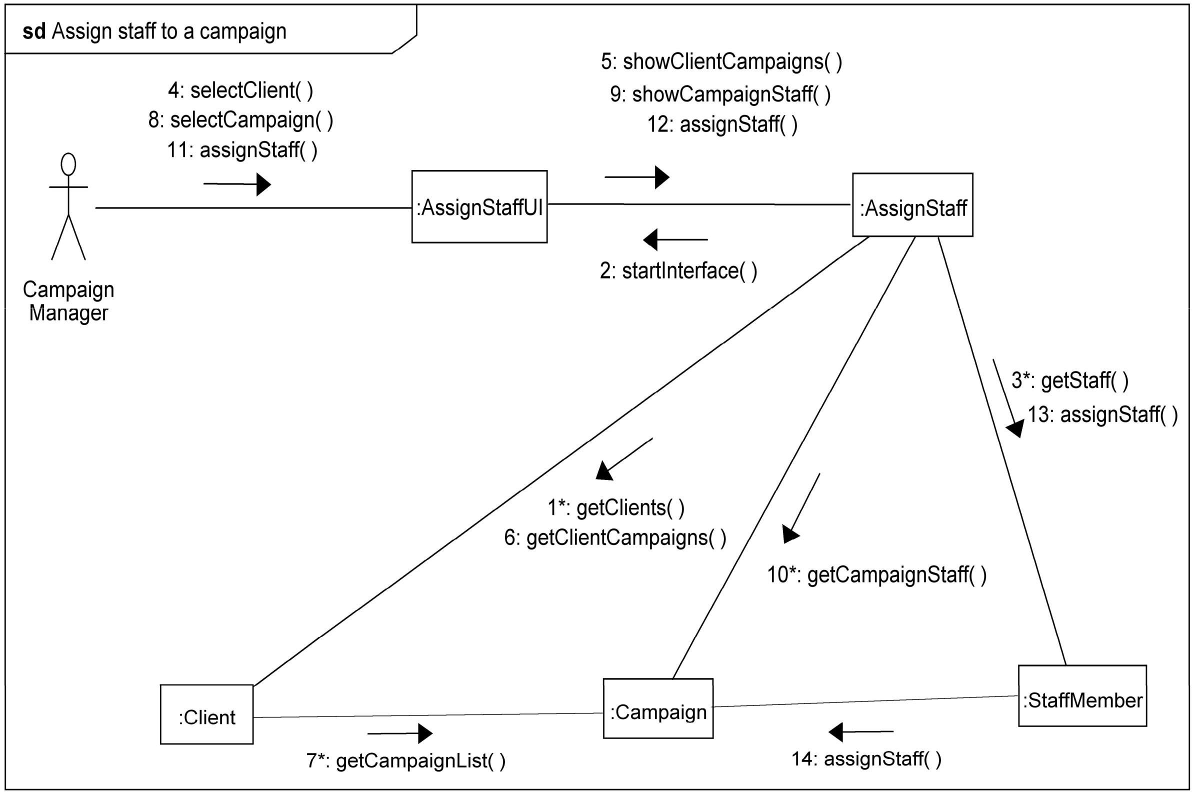
:Client

:Campaign

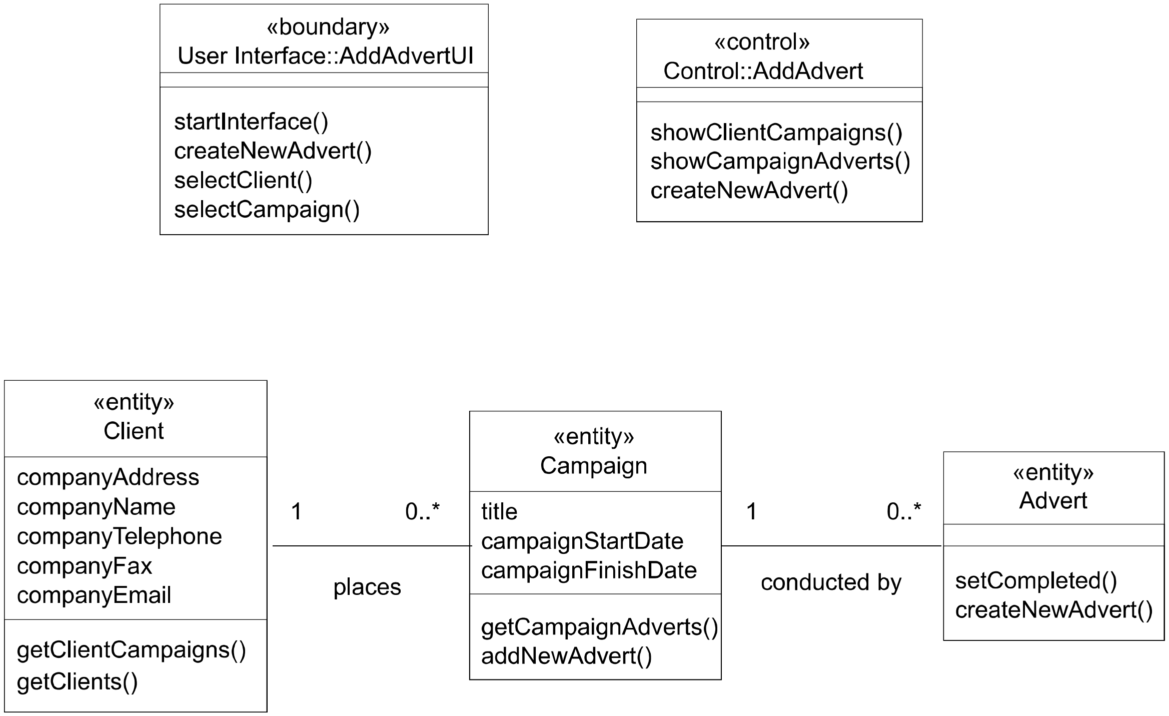
Early Draft Communication Diagram


:Campaign Manager
:AssignStaffUI
:AssignStaff
:Client
:Campaign
:StaffMember

More Developed Communication Diagram


sd Assign staff to a campaign
4: selectClient( )
8: selectCampaign( )
11: assignStaff( )
5: showClientCampaigns( )
9: showCampaignStaff( )
12: assignStaff( )
Campaign Manager
:AssignStaffUI
:AssignStaff
2: startInterface( )
3*: getStaff( )
13: assignStaff( )
1*: getClients( )
6: getClientCampaigns( )
10*: getCampaignStaff( )
:Client
:Campaign
:StaffMember
7*: getCampaignList( )
14: assignStaff( )

Resulting Class Diagram


«boundary»
User Interface::AddAdvertUI
startInterface()
createNewAdvert()
selectClient()
selectCampaign()
«control»
Control::AddAdvert
showClientCampaigns()
showCampaignAdverts()
createNewAdvert()
«entity»
Client
companyAddress
companyName
companyTelephone
companyFax
companyEmail
getClientCampaigns()
getClients()
1
0..*
places
«entity»
Campaign
title
campaignStartDate
campaignFinishDate
getCampaignAdverts()
addNewAdvert()
1
0..*
conducted by
«entity»
Advert
setCompleted()
createNewAdvert()

Reasonability Checks for Candidate Classes

- A number of tests help checks for candidate Class is reasonable
- Is it beyond the scope of the system ?
- Does it refer to the system as a whole ?
- Does it duplicate another class ?
- Is it too vague

Reasonability Checks for Candidate Classes

- Is it too tied up with physical inputs and outputs?
- Is it really an attribute?
- Is it really an operation?
- Is it really an association?
- If any of this is ‘Yes’, consider modelling the potential class in some other way(or do not model it at all)

CRC Cards

Class-Responsibility-Collaboration cards help to model interaction between objects

- Used as a way of:

- Identifying classes that participate in a scenario

- Allocating responsibilities – both operations and attributes (what can I do? and what do I know?)

- For a given scenario (or use case):

- Brainstorm the objects

- Allocate to team members

- Role play the interaction

CRC Cards

Class Name:	
Responsibilities	Collaborations
Responsibilities of a class are listed in this section.	*Collaborations with other classes are listed here, together with a brief description of the purpose of the collaboration.*

Class Name	Client
Responsibilities	Collaborations
Provide client information.	
Provide list of campaigns.	Campaign provides campaign details.

Class Name	Campaign
Responsibilities	Collaborations
Provide campaign information.	
Provide list of adverts.	Advert provides advert details.
Add a new advert.	Advert constructs new object.

Class Name	Advert
Responsibilities	Collaborations
Provide advert details.	
Construct adverts.	

CRC Cards

- Effective role play depends on an explicit strategy for distributing responsibility among classes

- For example:

- Each role player tries to be lazy

- Persuades other players their class should accept responsibility for a given task

- May use ‘Paper CASE’ to document the associations and links

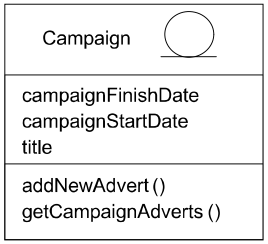
Assembling Class Diagram

However individual use cases are analysed, the aim is to produce a single analysis class diagram

This models the application as a whole

The concept is simple:

- A class is the analysis model needs all the details required for that class in each separate use case

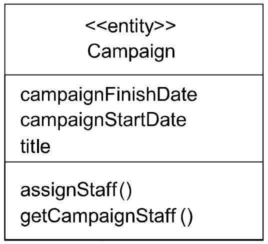


Campaign

campaignFinishDate
campaignStartDate
title

addNewAdvert ()
getCampaignAdverts ()

(a) Campaign class that meets the needs of `Add new advert to a campaign`



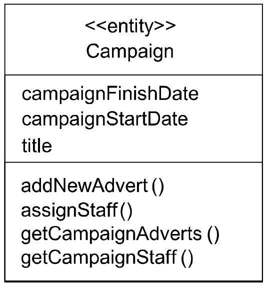
<<entity>>
Campaign

campaignFinishDate
campaignStartDate
title

assignStaff ()
getCampaignStaff ()

(b) Campaign class that meets the needs of `Assign staff to work on a campaign`

(c) Campaign class that meets the needs of both use cases

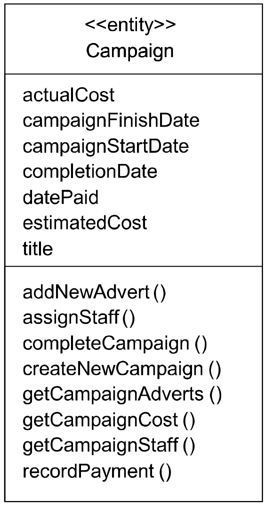


<<entity>>
Campaign

campaignFinishDate
campaignStartDate
title

addNewAdvert ()
assignStaff ()
getCampaignAdverts ()
getCampaignStaff ()

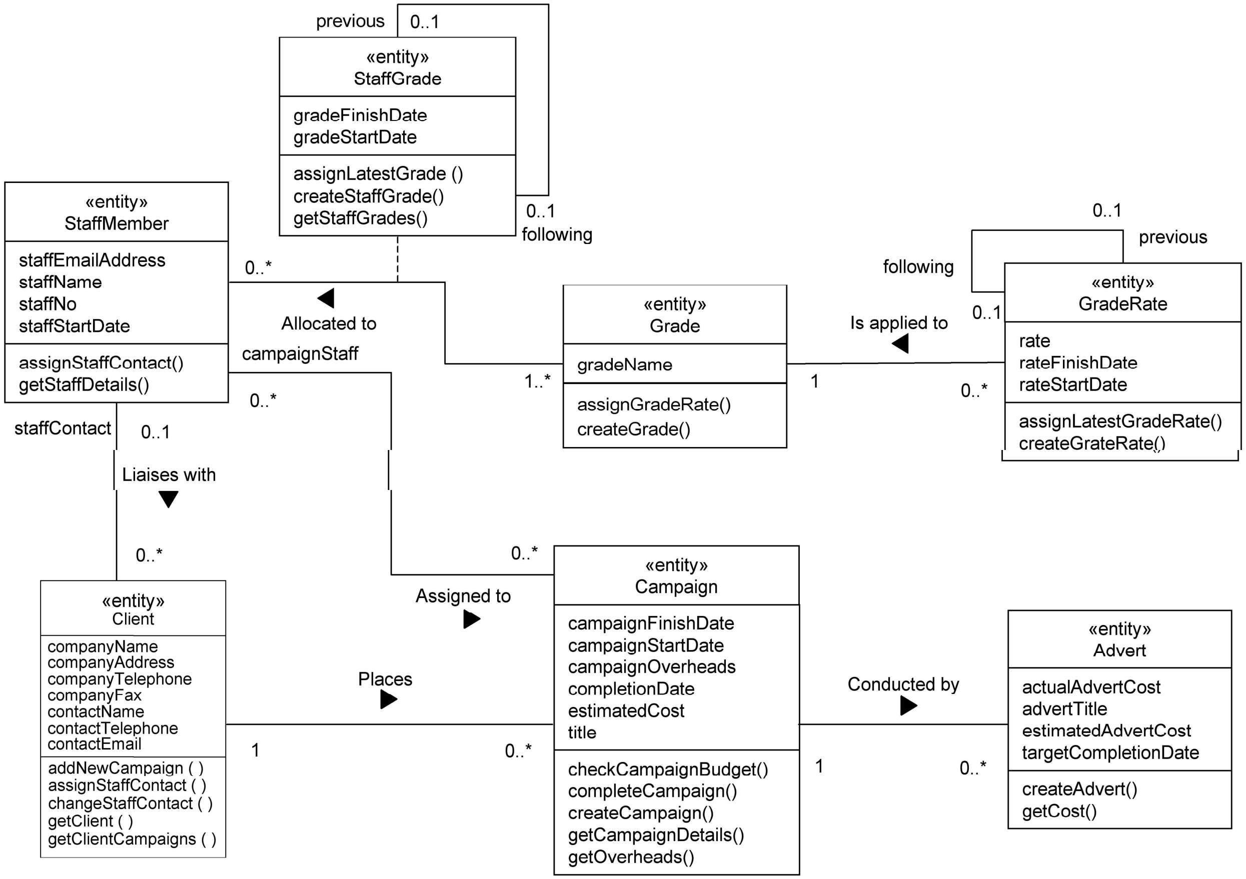
(d) A more fully developed Campaign class meets the requirements of these and several other use cases too



<<entity>>
Campaign

actualCost
campaignFinishDate
campaignStartDate
completionDate
datePaid
estimatedCost
title

addNewAdvert ()
assignStaff ()
completeCampaign ()
createNewCampaign ()
getCampaignAdverts ()
getCampaignCost ()
getCampaignStaff ()
recordPayment ()



previous 0..1

«entity»
StaffGrade

gradeFinishDate
gradeStartDate

assignLatestGrade ()
createStaffGrade()
getStaffGrades()

0..1
following

«entity»
StaffMember

staffEmailAddress
staffName
staffNo
staffStartDate

assignStaffContact()
getStaffDetails()

0..*
Allocated to
campaignStaff
0..*
1..*

«entity»
Grade

gradeName

assignGradeRate()
createGrade()

1
Is applied to
0..*
following
0..1
0..1
previous

«entity»
GradeRate

rate
rateFinishDate
rateStartDate

assignLatestGradeRate()
createGrateRate()

staffContact
0..1
Liaises with
0..*

«entity»
Client

companyName
companyAddress
companyTelephone
companyFax
contactName
contactTelephone
contactEmail

addNewCampaign ( )
assignStaffContact ( )
changeStaffContact ( )
getClient ( )
getClientCampaigns ( )

Assigned to
0..*
Places
1
0..*

«entity»
Campaign

campaignFinishDate
campaignStartDate
campaignOverheads
completionDate
estimatedCost
title

checkCampaignBudget()
completeCampaign()
createCampaign()
getCampaignDetails()
getOverheads()

Conducted by
1
0..*

«entity»
Advert

actualAdvertCost
advertTitle
estimatedAdvertCost
targetCompletionDate

createAdvert()
getCost()

System Specification

- Goal: understand (and specify) what the system should do
- System Sequence Diagram
- What are the operations on the system, performed by the user?
- Examples: “run the program with input I”, “Select menu command C”
- In what order do they occur?
- Must compile a file before running it

Behavioral Contracts

- Under what conditions can each interaction take place?
- A file must typecheck correctly before you can compile it
- What is the result of the interaction?
- a .class file is created
- The .class file is named after the class in the source code
- …

A System Sequence Diagram is a picture that shows,

for one scenario of use, who interacts with the system,

and the sequence of events that occur on the system’s boundary

e.g. use case: A commit and pull request using GitHub (aside: distributed VCS for large projects)

1. Contributor modifies source code, commits changes to her own fork of the GitHub repository.

2. GitHub applies commit to the fork's history and acknowledges the commit.

3. Contributor initiates a pull request, describing the changes (he/She) has committed.

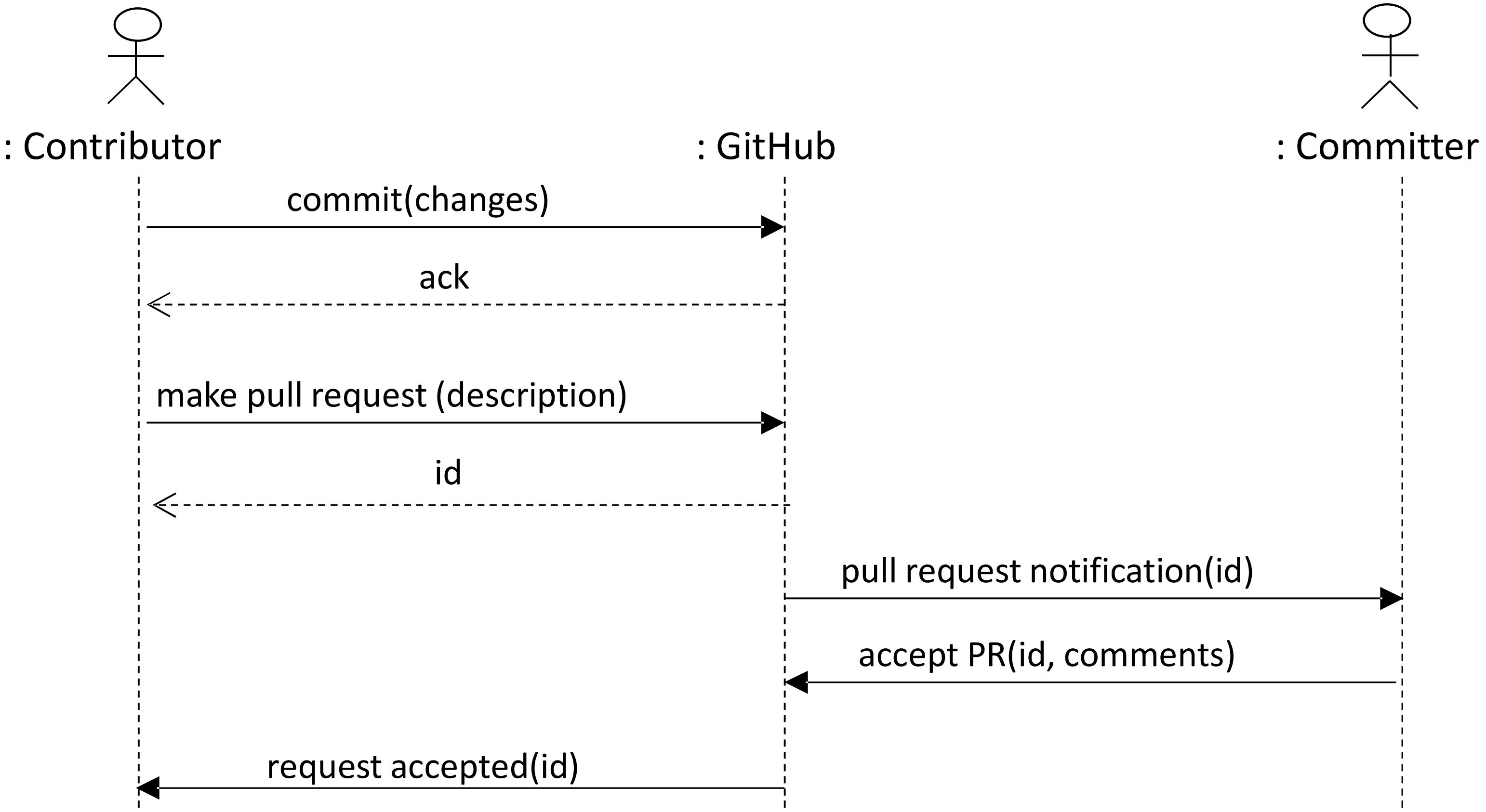
4. GitHub acknowledges the pull request and returns an ID for the pull request, and notifies the Committer of the pull request.

5. Committer inspects Contributor's work and accepts the pull request into the main repository, notifying GitHub of the accepted pull request and any comments associated with the acceptance.

6. GitHub notifies the Contributor that her work was accepted.

System Sequence Diagram

A System Sequence Diagram is a picture that shows, for one scenario of use, who interacts with the system, and the sequence of events that occur on the system's boundary


: Contributor
: GitHub
: Committer
commit(changes)
ack
make pull request (description)
id
pull request notification(id)
accept PR(id, comments)
request accepted(id)

Sequence Diagram for the Point of Sale Example

Use Case: Successful Customer Checkout

1. Customer arrives at POS checkout with goods to purchase
2. Cashier starts a new sale
3. Cashier enters item identifier and quantity
4. System records sale line item and presents item description, price, and running total

Sequence Diagram for the Point of Sale Example

5. Cashier repeats steps 3-4 until all goods have been entered

6. System presents total with taxes calculated

7. Cashier tells customer the total, and asks for payment

8. Customer pays and System provides change and a receipt

Behavioral Contracts: What do These Operations Do?

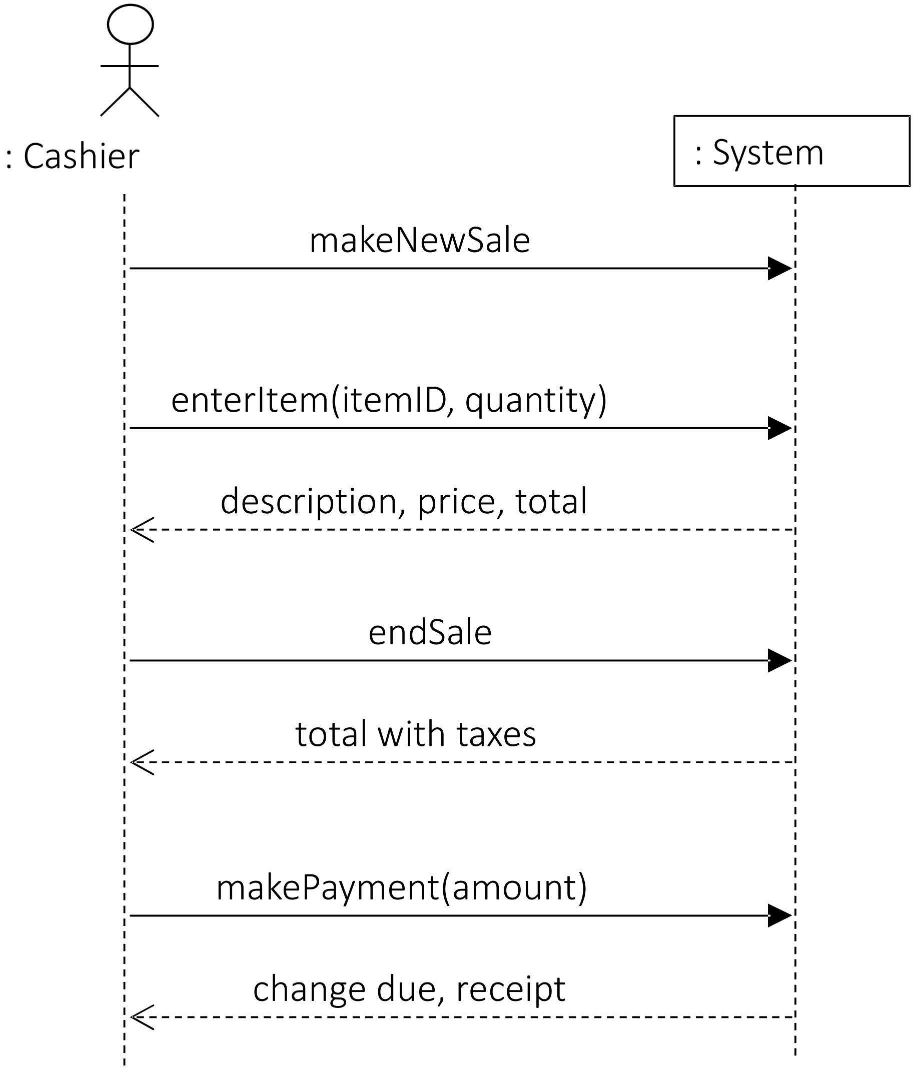
To design, we need a spec

- Preconditions
- Postconditions
- We can write a behavioral contract
- Like a pre-/post-condition

Behavioral Contracts: What do These Operations Do?

- specification for code
- Often written in natural language
- Focused on system interfaces
- may or may not be methods

Sequence Diagram for the Point of Sale Example



: Cashier

: System

makeNewSale

enterItem(itemID, quantity)

description, price, total

endSale

total with taxes

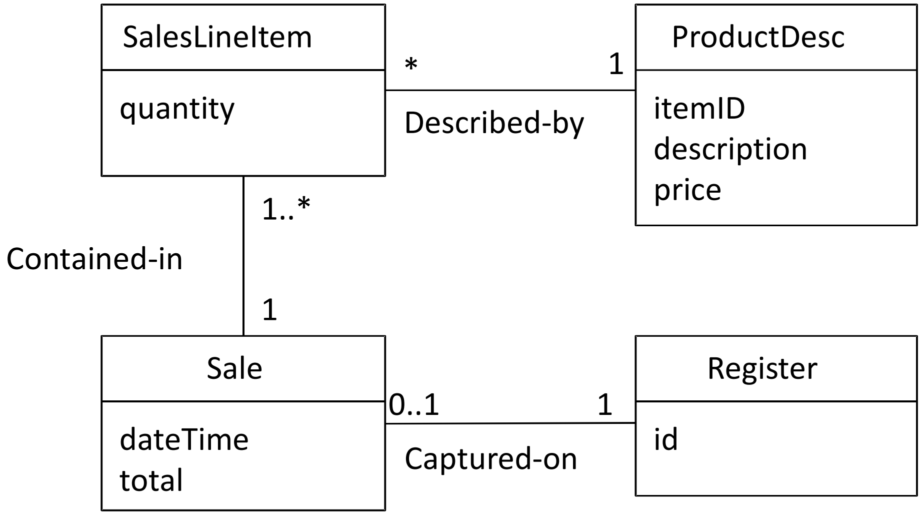
makePayment(amount)

change due, receipt

Example Point of Sale Contract

Operation: makeNewSale()
Preconditions: There is not currently a sale in progress

Postconditions:
- A Sale instance s was created
- s was associated with a Register


SalesLineItem
quantity
ProductDesc
itemID
description
price
*
1
Described-by
1..*
Contained-in
1
Sale
dateTime
total
Register
id
0..1
1
Captured-on

A Point of Sale Contract

- Contract structure
  - Operation name, parameters
  - Requirement or use case this is a part of
  - Preconditions
  - Postconditions

Operation: makeNewSale()

Preconditions: There is not currently a sale in progress

Postconditions:
- A Sale instance s was created
- s was associated with a Register

- Which contracts to write?
  - Operations that are complex or subtle
  - Operations that not everyone understands
  - Simple/obvious operations are often not given contracts in practice

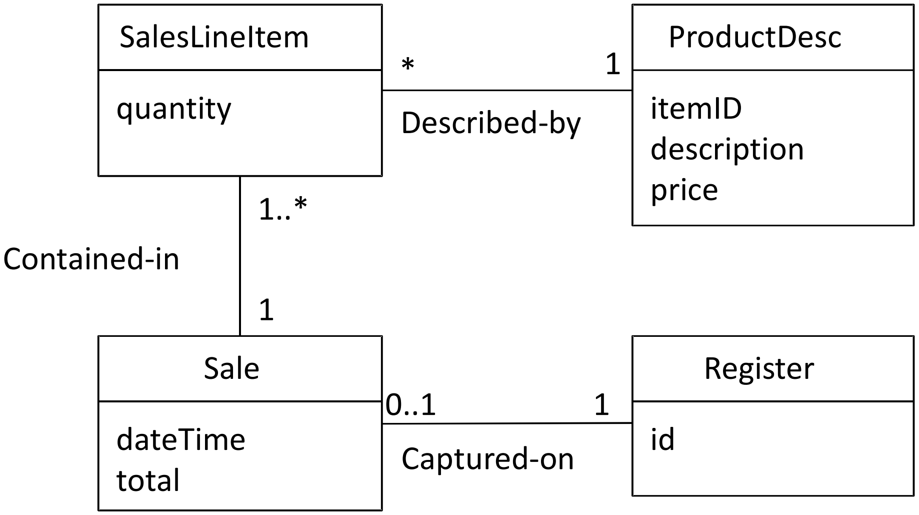
A Point of Sale Contract

- Writing postconditions
  Written in past tense (a post-condition)
  Describe changes to domain model
    - Instance creation and deletion
    - Attribute modification
    - Associations formed and broken
      - Easy to forget associations when creating objects!

Operation: enterItem(itemID : ItemID, quantity : integer)

Preconditions:

Postconditions:



SalesLineItem

quantity

ProductDesc

itemID
description
price

Described-by

* 1

Contained-in

1..* 1

Sale

dateTime
total

Register

id

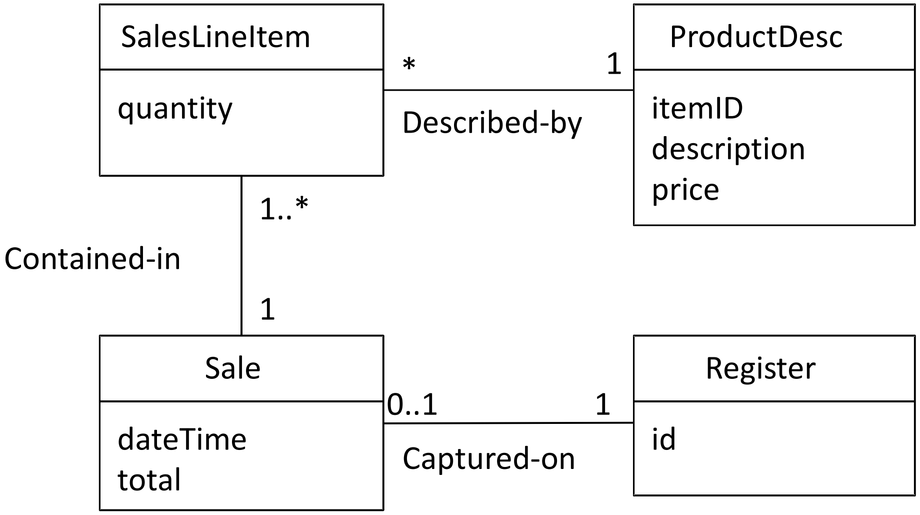
Captured-on

0..1 1

Operation: makeNewSale()

Preconditions: There is not currently a sale in progress

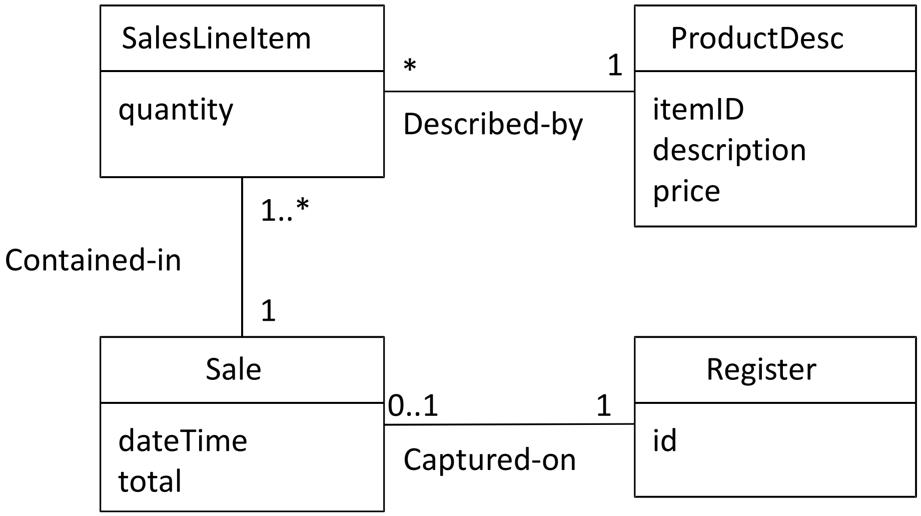
Postconditions:
- A Sale instance s was created
- s was associated with a Register


SalesLineItem
quantity
*
Described-by
1
ProductDesc
itemID
description
price
1..*
Contained-in
1
Sale
dateTime
total
0..1
Captured-on
1
Register
id

Operation: enterItem(itemID : ItemID, quantity : integer)

Preconditions: There is a sale s in progress

Postconditions:
- A SalesLineItem instance sli was created
- sli was associated with the sale s
- sli.quantity became quantity
- sli was associated with a ProductDescription, based on itemID match


SalesLineItem
quantity
*
1
Described-by
ProductDesc
itemID
description
price
1..*
Contained-in
1
Sale
dateTime
total
0..1
1
Captured-on
Register
id

Responsibilities

obligations that an object has to fulfill

Two types of responsibilities:

knowing
doing

Knowing responsibilities of an object include:

- knowing about private encapsulated data
- knowing about related objects
- knowing about things it can derive or calculate

Responsibilities

Doing responsibilities of an object include:

- doing something itself, such as creating an object or doing a calculation
- initiating action in other objects
- controlling and coordinating activities in other objects

Responsibility Assignment

- Consider the system’s operations

    Found in System Sequence Diagrams and Behavioral Contracts

- Divide each system operation into responsibilities

    Knowing, doing, etc.

- Assign each responsibility to an object

    Knowing

    - Domain-related “knowing” responsibilities already in domain model
    - Implementation-related “knowing” responsibilities will be new

    Doing

    - Become methods in the implementation

Responsibility Assignment

- Allocate object responsibilities to classes

  Two objects may have the same class merge responsibilities

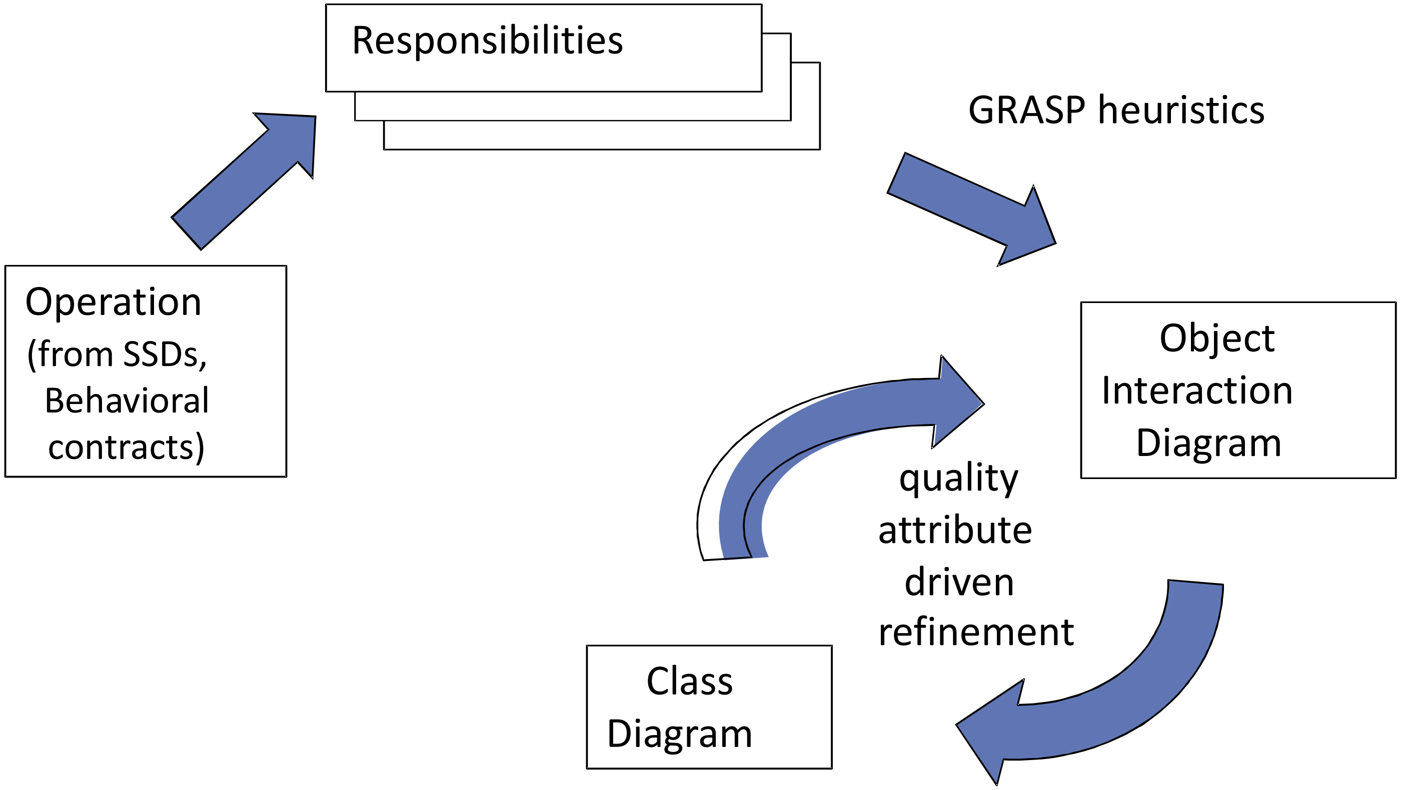
  A class may have a superclass divide responsibilities

- How to assign

  Drive by quality attributes

  General Responsibility Assignment Software Patterns provide heuristics (GRASP patterns for short)

Responsibility Assignment, Graphically



Responsibilities

GRASP heuristics

Operation (from SSDs, Behavioral contracts)

Object Interaction Diagram

quality attribute driven refinement

Class Diagram

Sequence diagram – Iteration, conditional interactions

Importance of Sequence Diagrams

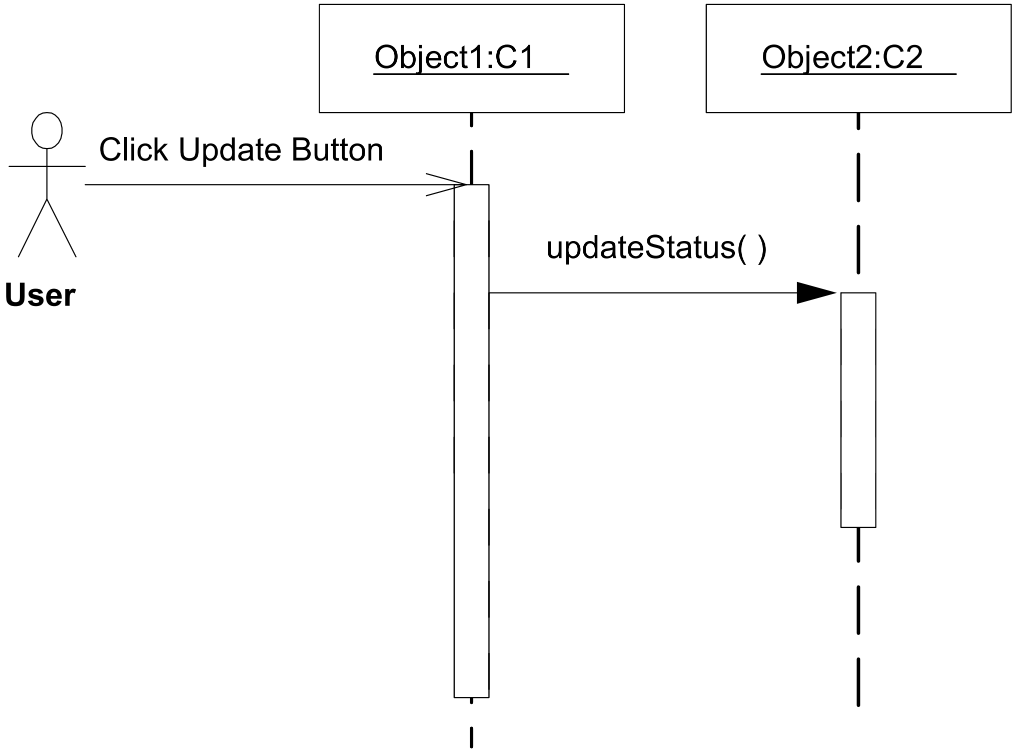
- Depict object interactions in a given scenario identified for a given Use Case

- Specify the messages passed between objects using horizontal arrows including messages to/from external actors

- Time increases from Top to bottom

Rules of Sequence Diagrams

- Sequence Initiation


Object1:C1
Object2:C2
Click Update Button
User
updateStatus( )

Rules of Sequence Diagrams

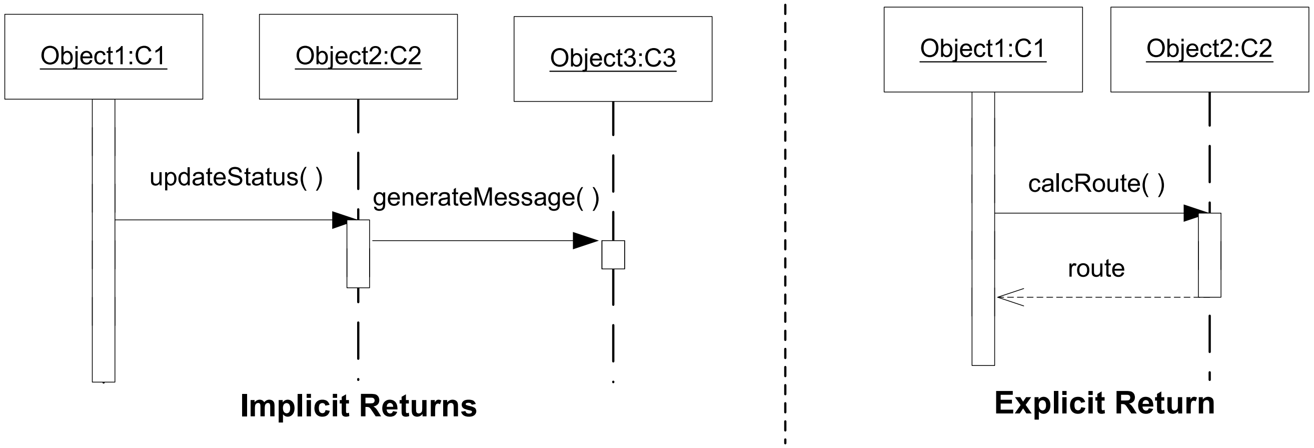
- Identify objects needed to support use case, determine sequence of internal events following the external initiating event

- Diagrams that are not initiated with an external actor represent only a partial sequence

- Partial sequence diagrams should clearly identify the actor initiated sequence diagrams from which they are launched

Rules of Sequence Diagrams

- Messages specified on interactions can be synchronous or asynchronous


Object1:C1
Object2:C2
Object3:C3
updateStatus( )
generateMessage( )
Object1:C1
Object2:C2
calcRoute( )
route

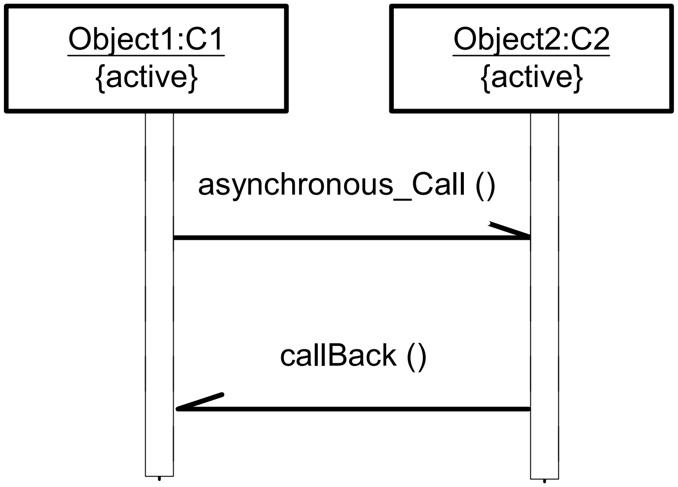
Implicit Returns

Explicit Return

Synchronous call

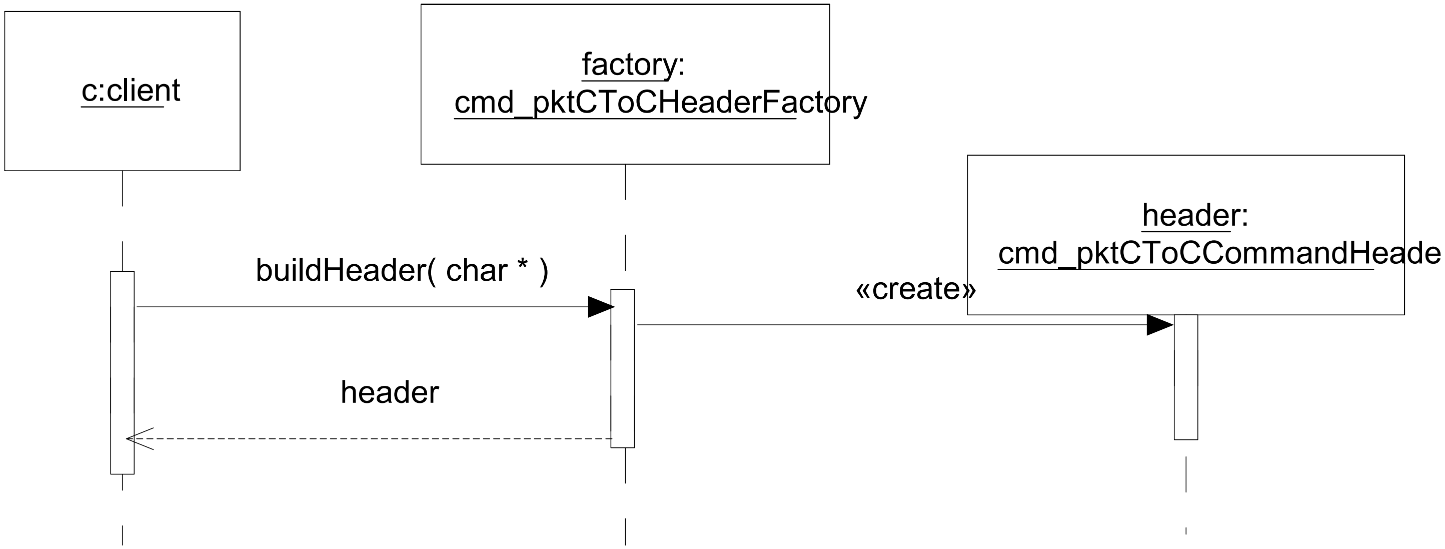
Rules of Sequence Diagrams

Asynchronous call


Object1:C1
{active}
Object2:C2
{active}
asynchronous_Call ()
callBack ()

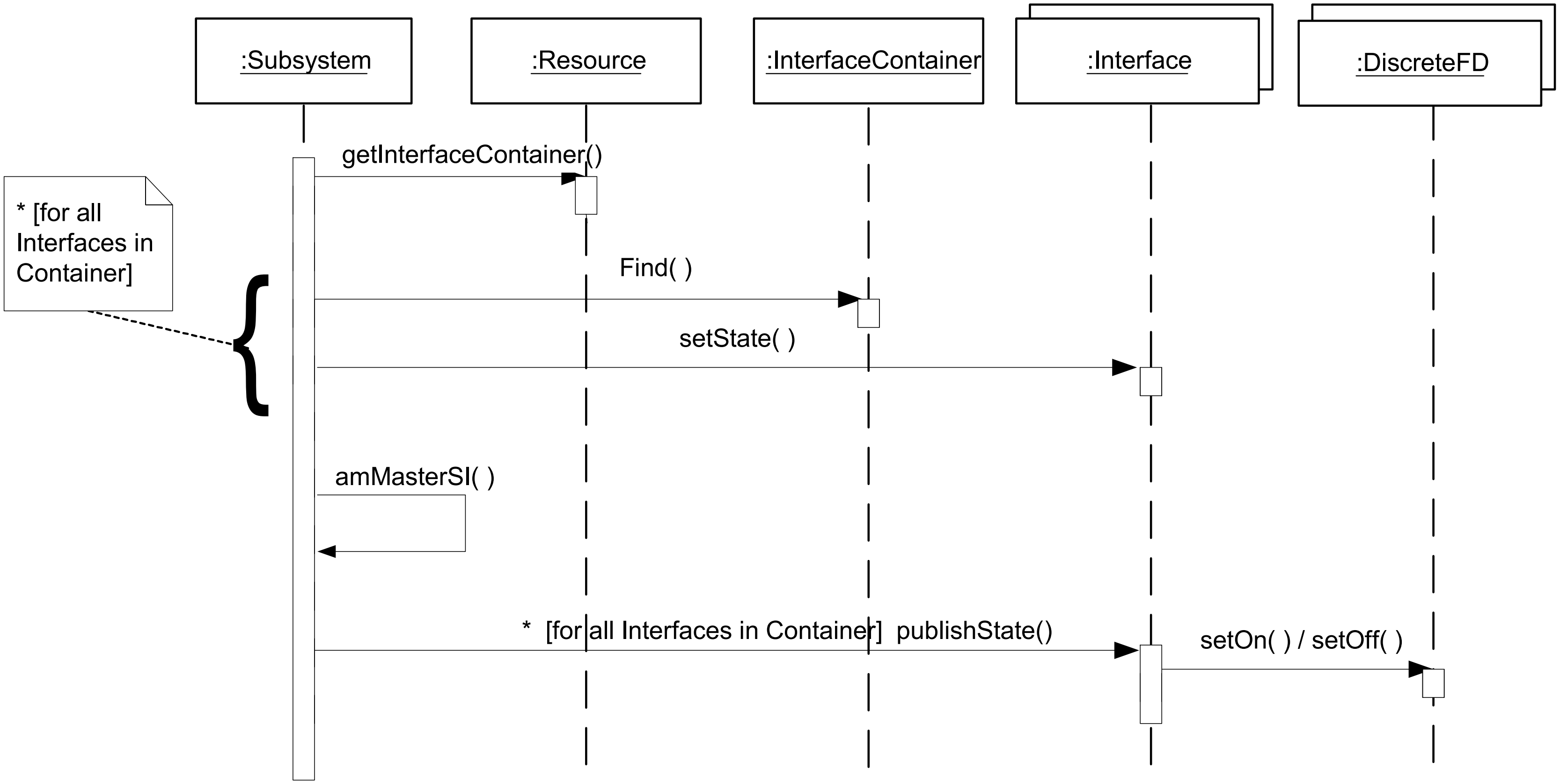
Rules of Sequence Diagrams

- Display operation names on call arrows

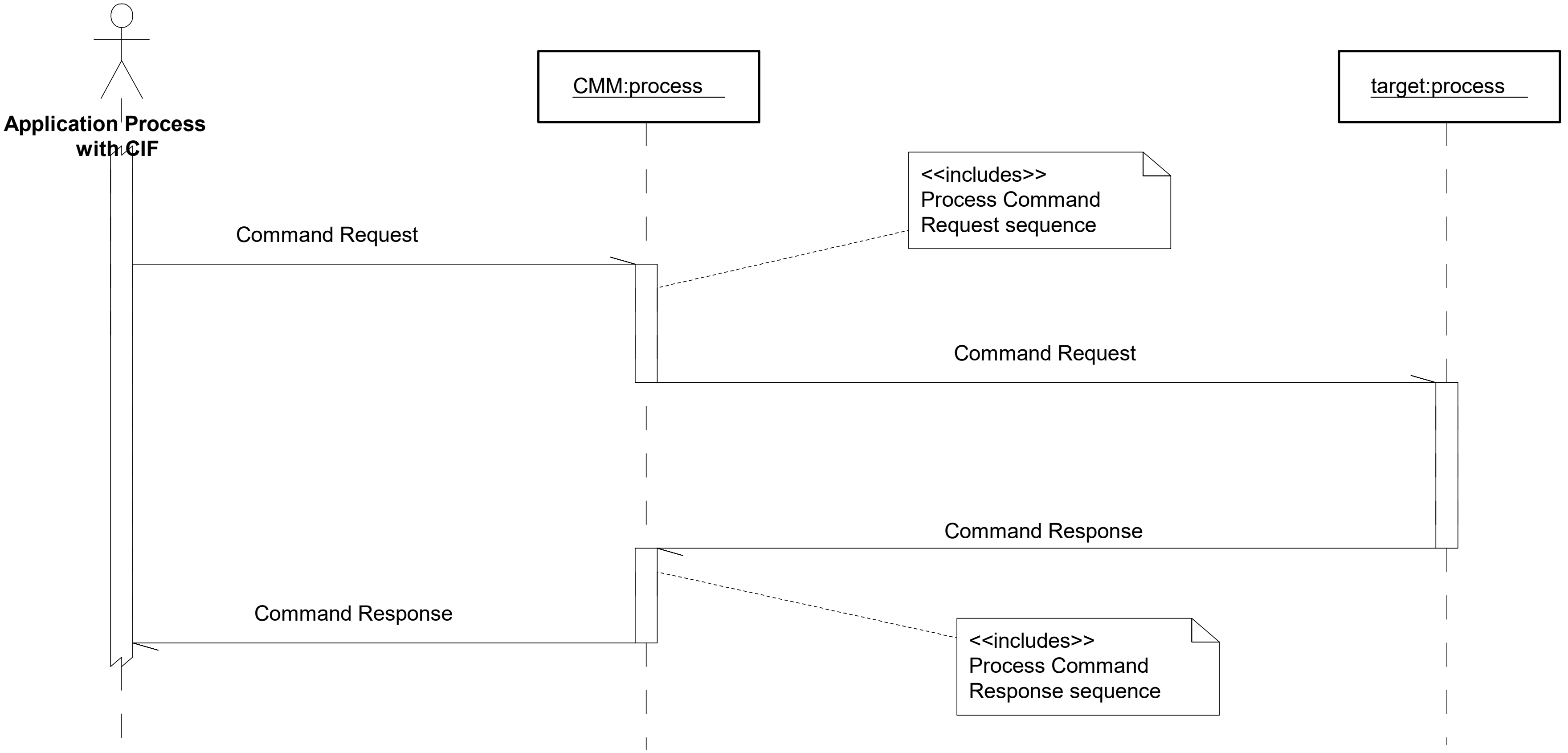

c:client
factory:
cmd_pktCToCHeaderFactory
header:
cmd_pktCToCCommandHeade
buildHeader(char *)
«create»
header

Rules of Sequence Diagrams

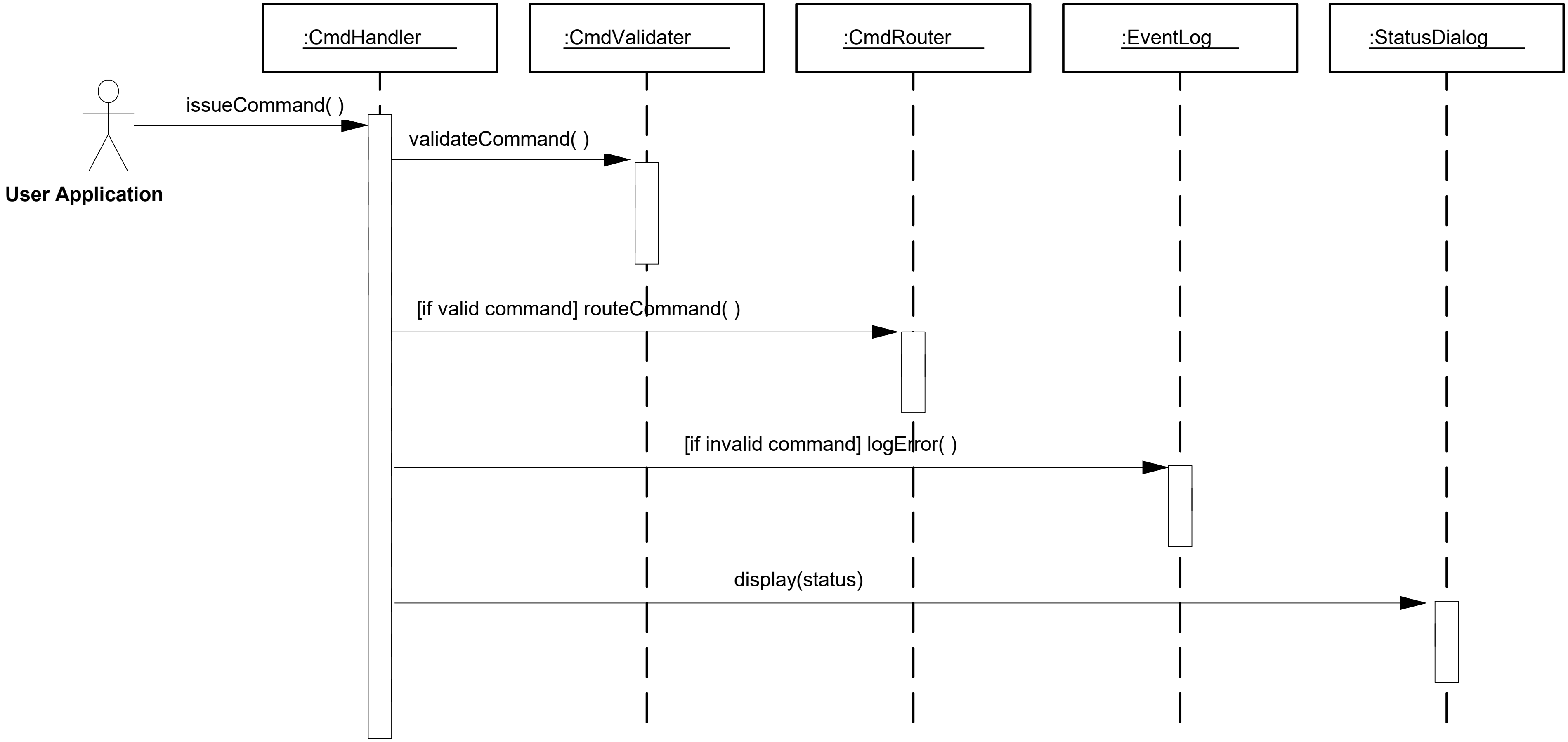
Compound and Simple Iteration


:Subsystem
:Resource
:InterfaceContainer
:Interface
:DiscreteFD
getInterfaceContainer()
* [for all Interfaces in Container]
Find( )
setState( )
amMasterSI( )
* [for all Interfaces in Container] publishState()
setOn( ) / setOff( )

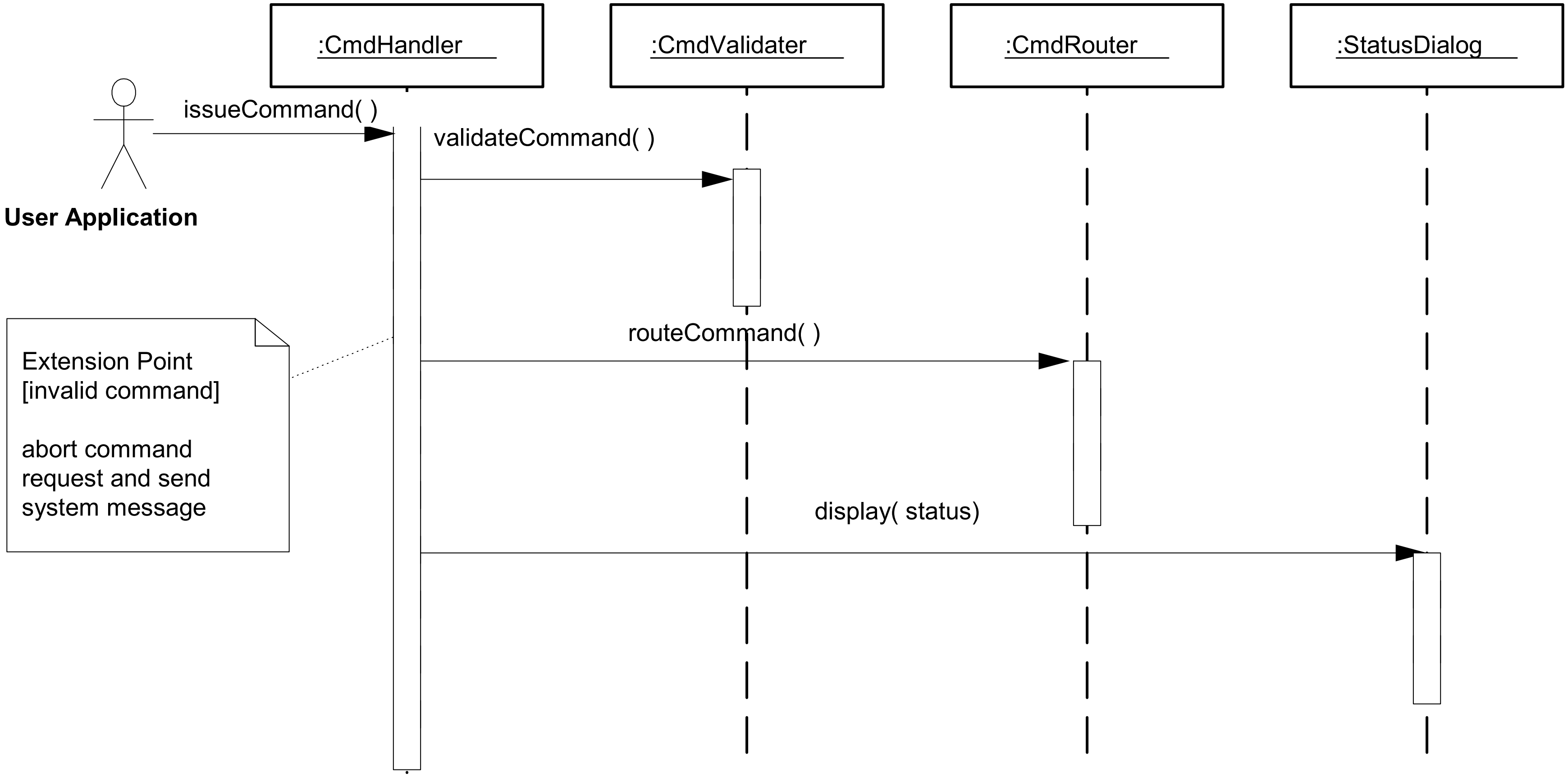
‘included’ sequence diagrams


Application Process with CIF
CMM:process
target:process
<<includes>> Process Command Request sequence
Command Request
Command Request
Command Response
Command Response
<<includes>> Process Command Response sequence

Showing alternate behavior in a sequence diagram


:CmdHandler
:CmdValidater
:CmdRouter
:EventLog
:StatusDialog
issueCommand( )
User Application
validateCommand( )
[if valid command] routeCommand( )
[if invalid command] logError( )
display(status)

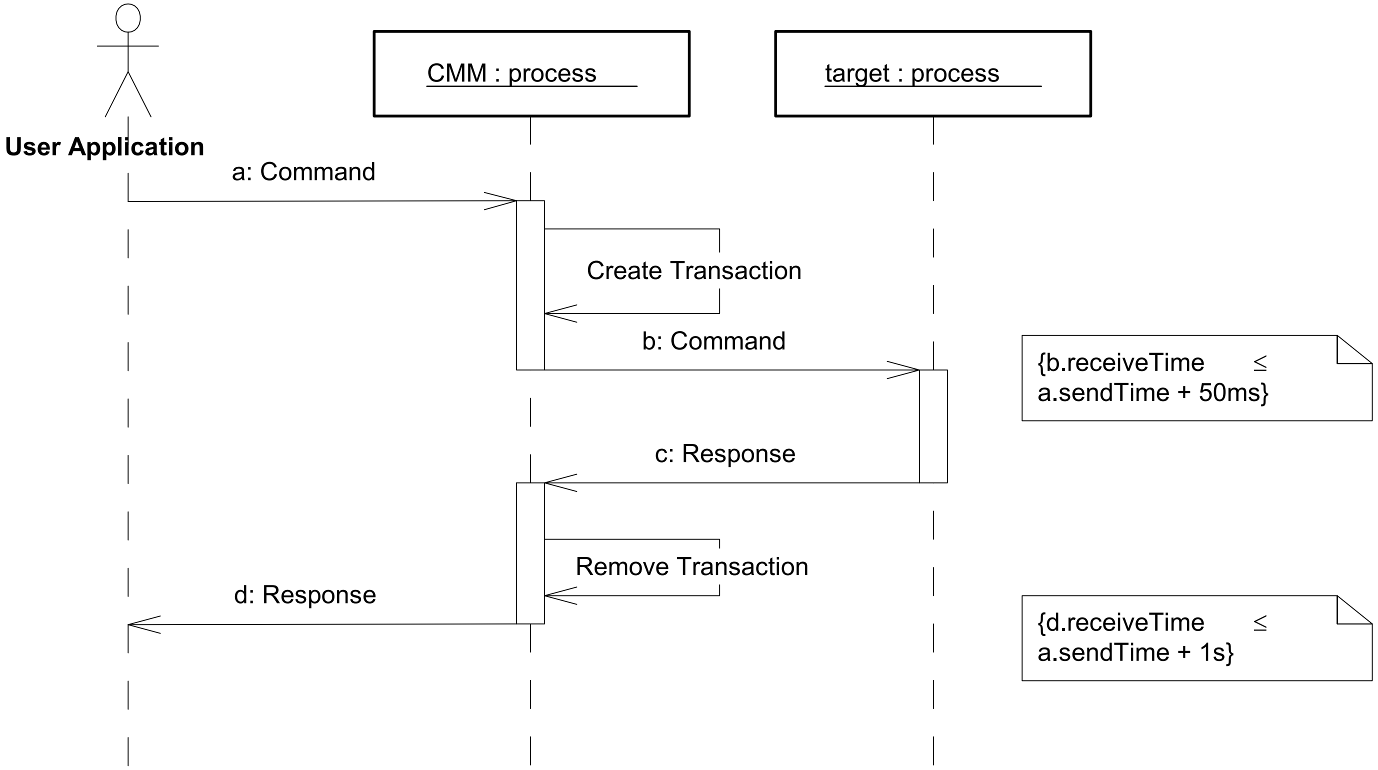
Showing Extension Point


:CmdHandler
:CmdValidater
:CmdRouter
:StatusDialog
issueCommand( )
User Application
validateCommand( )
routeCommand( )
display(status)

Extension Point
[invalid command]

abort command
request and send
system message

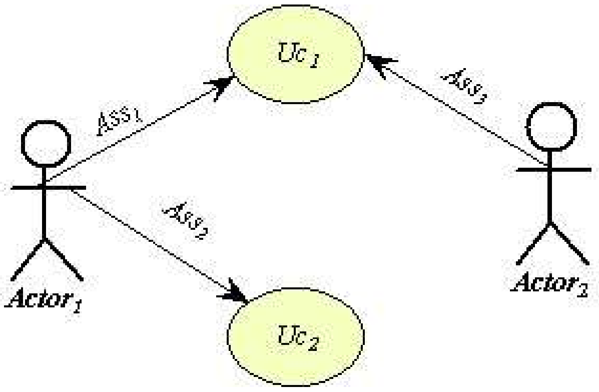
Specifying Timing Requirements


User Application
CMM : process
target : process
a: Command
Create Transaction
b: Command
c: Response
Remove Transaction
d: Response
{b.receiveTime ≤ a.sendTime + 50ms}
{d.receiveTime ≤ a.sendTime + 1s}

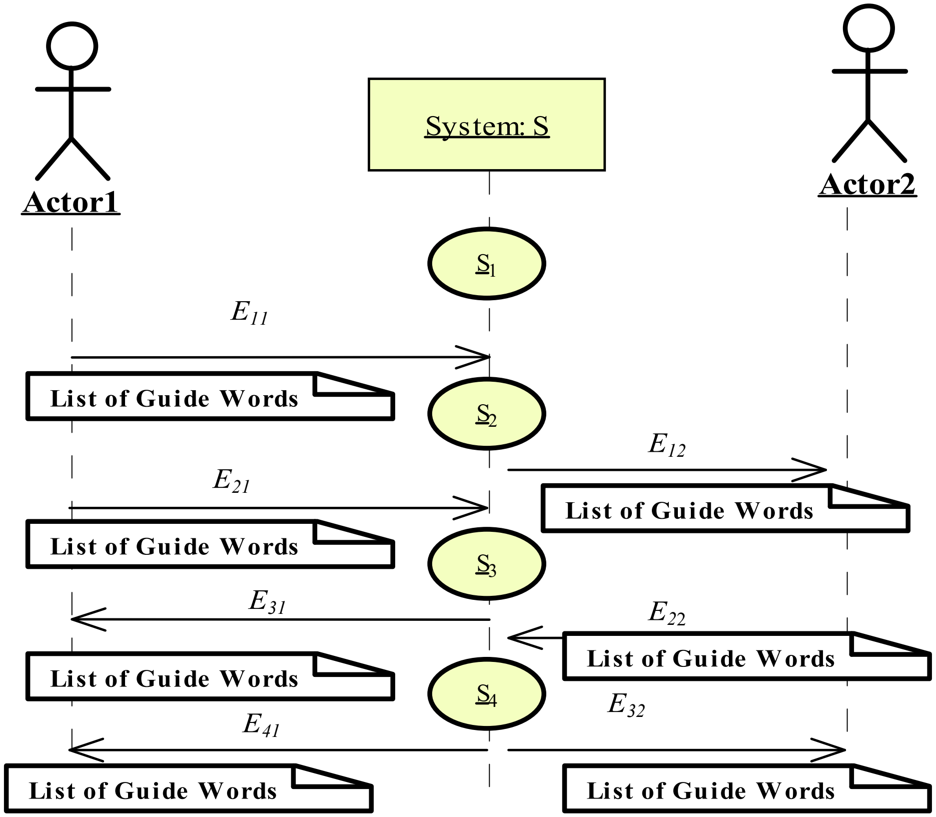
Requirements Elicitation Process- Step 4. Refining Use Cases using System Sequence Diagrams

- System sequence diagrams establish the dynamic behavior in terms of key scenarios of the system for each use case
- The system sequence diagram models a scenario of the system interactions with the environment for a given use case
- Input/output events are clearly identified in each sequence diagram,
- The State of the system before and after each event are also depicted
- Different diagrams model scenarios with the normal flow of events and the abnormal flow of events

Sequence Diagrams and Use Cases
System Sequence Diagram

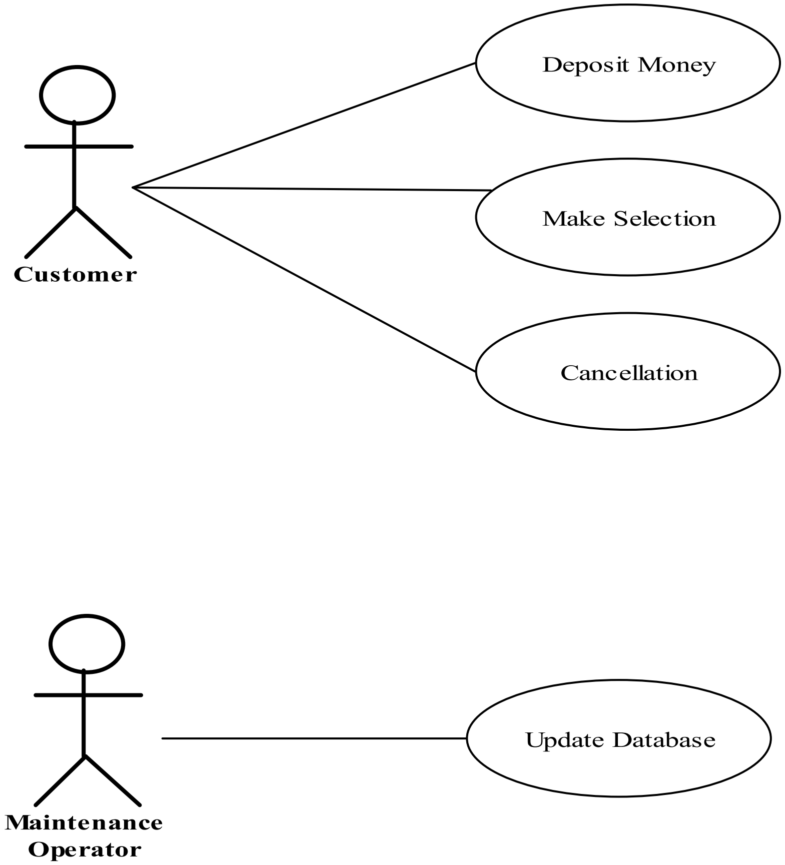

Uc 1
Ass 1
Ass 3
Ass 2
Actor 1
Actor 2
Uc 2

The use case diagram
Of system S

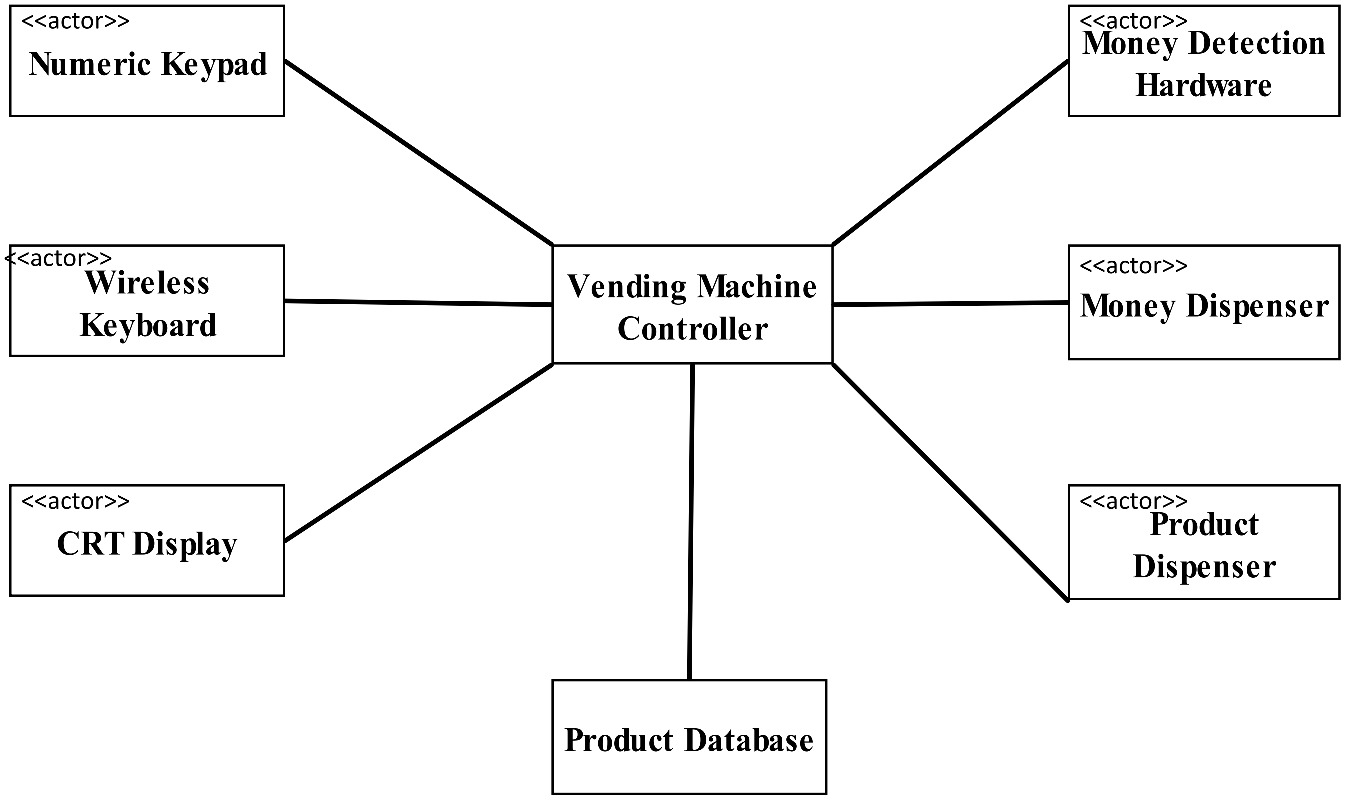

Actor1
System: S
Actor2
S1
E11
List of Guide Words
S2
E12
E21
List of Guide Words
List of Guide Words
S3
E31
E22
List of Guide Words
List of Guide Words
S4
E32
E41
List of Guide Words
List of Guide Words

The sequence diagram of use case UC1 for system S

Example: Use Case Diagram of the Vending Machine


Deposit Money
Make Selection
Cancellation
Customer
Update Database
Maintenance Operator

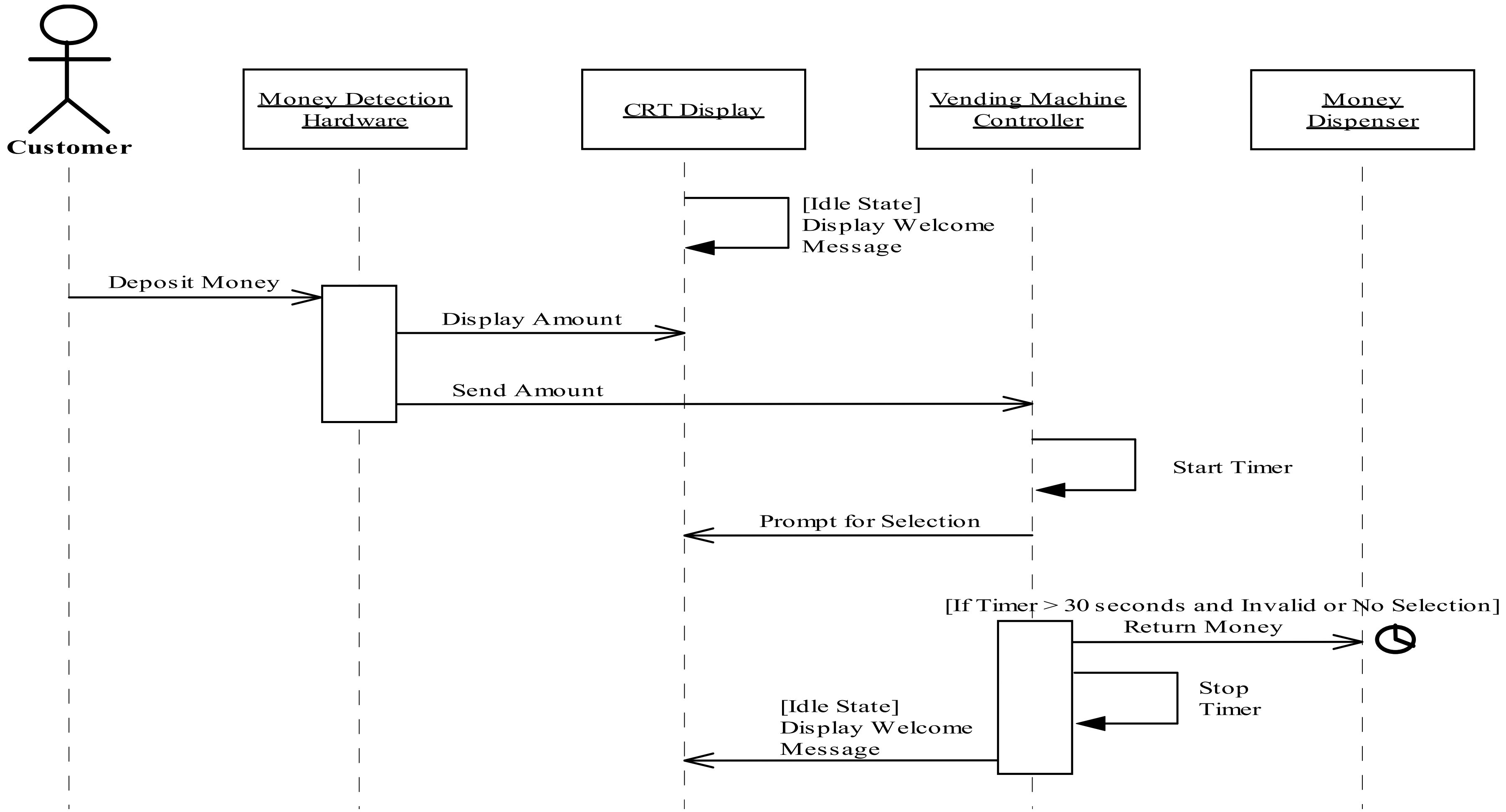
Requirements Elicitation Process
Step 6. Identify Initial Analysis Objects- The Initial Object-Model



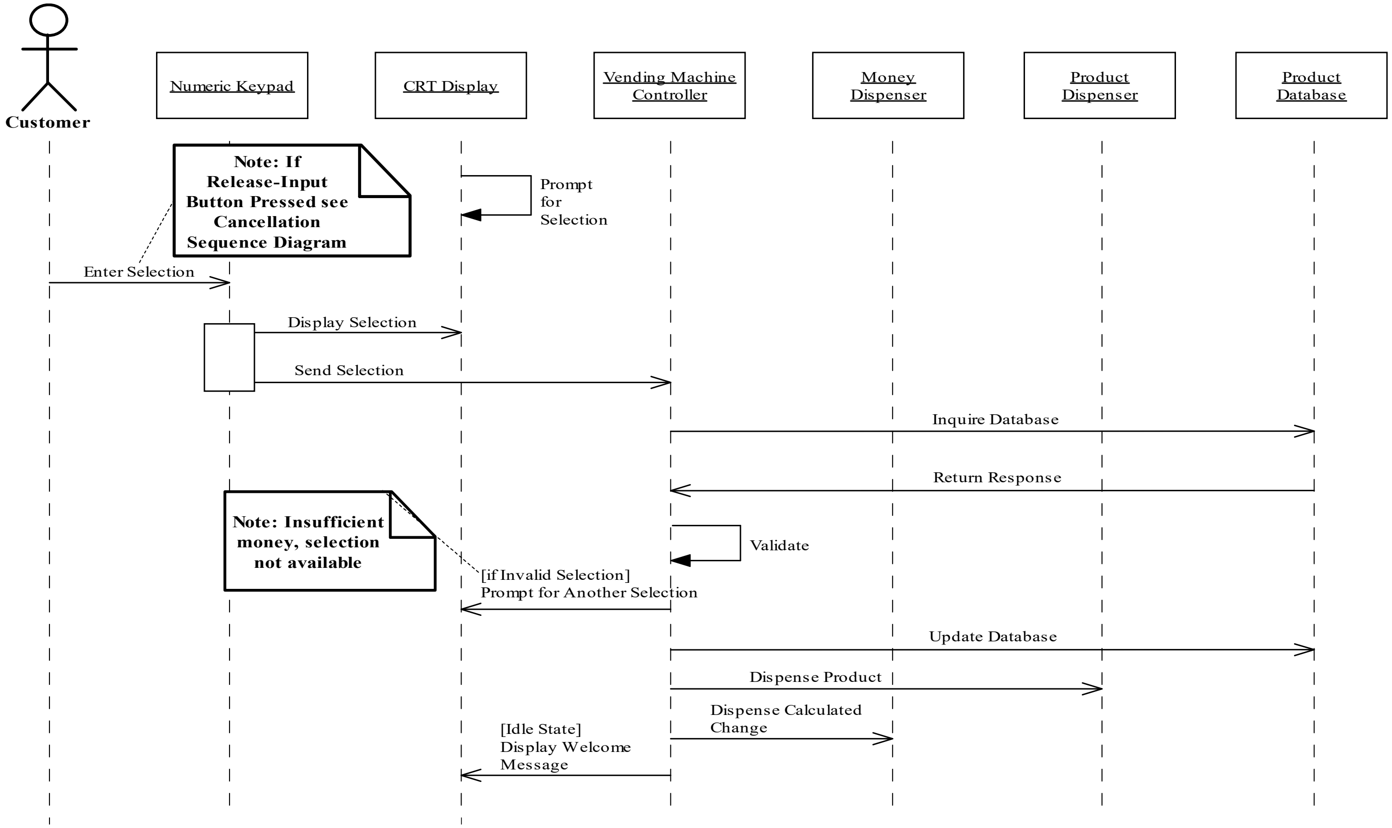
<<actor>>
Numeric Keypad
<<actor>>
Wireless Keyboard
<<actor>>
CRT Display
Vending Machine Controller
Product Database
<<actor>>
Money Detection Hardware
<<actor>>
Money Dispenser
<<actor>>
Product Dispenser

Designate the Actor objects with the <<actor>> stereotype above the object label

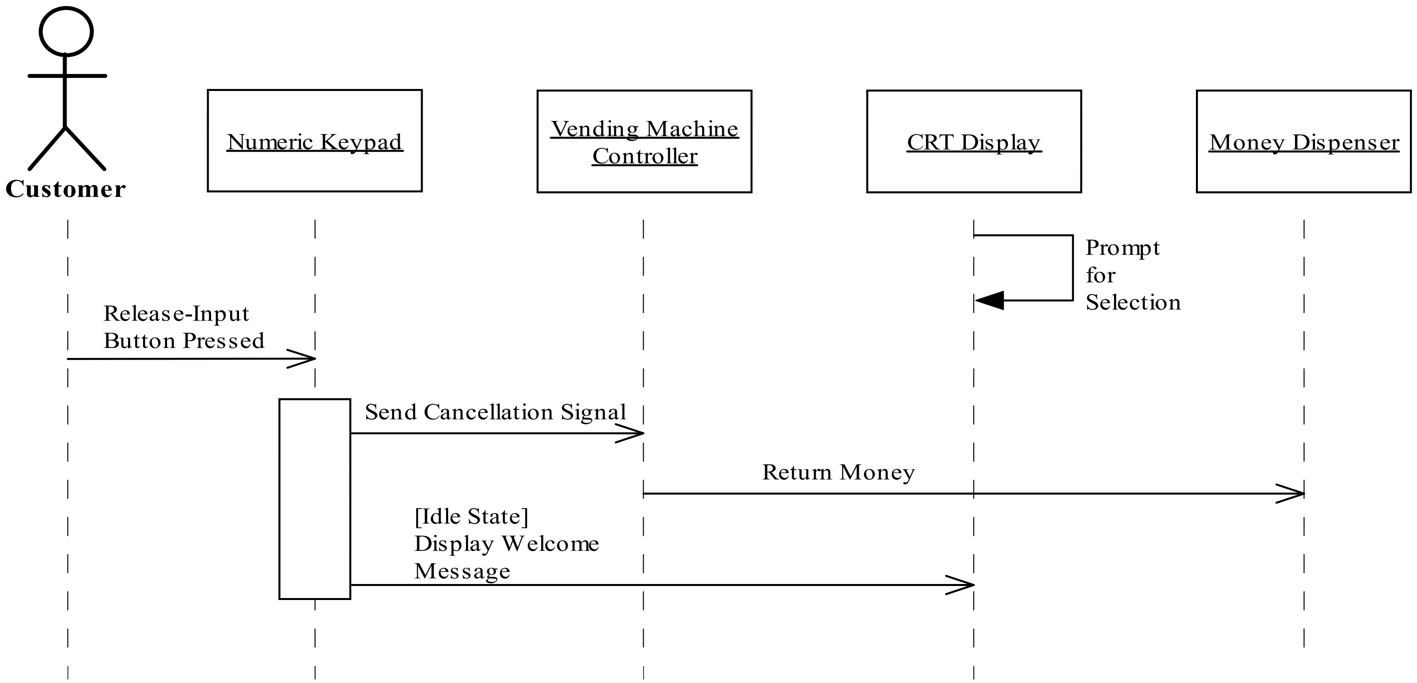
System Sequence Diagram for Deposit Money Use Case


Customer
Money Detection Hardware
CRT Display
Vending Machine Controller
Money Dispenser
[Idle State] Display Welcome Message
Deposit Money
Display Amount
Send Amount
Start Timer
Prompt for Selection
[If Timer > 30 seconds and Invalid or No Selection] Return Money
Stop Timer
[Idle State] Display Welcome Message

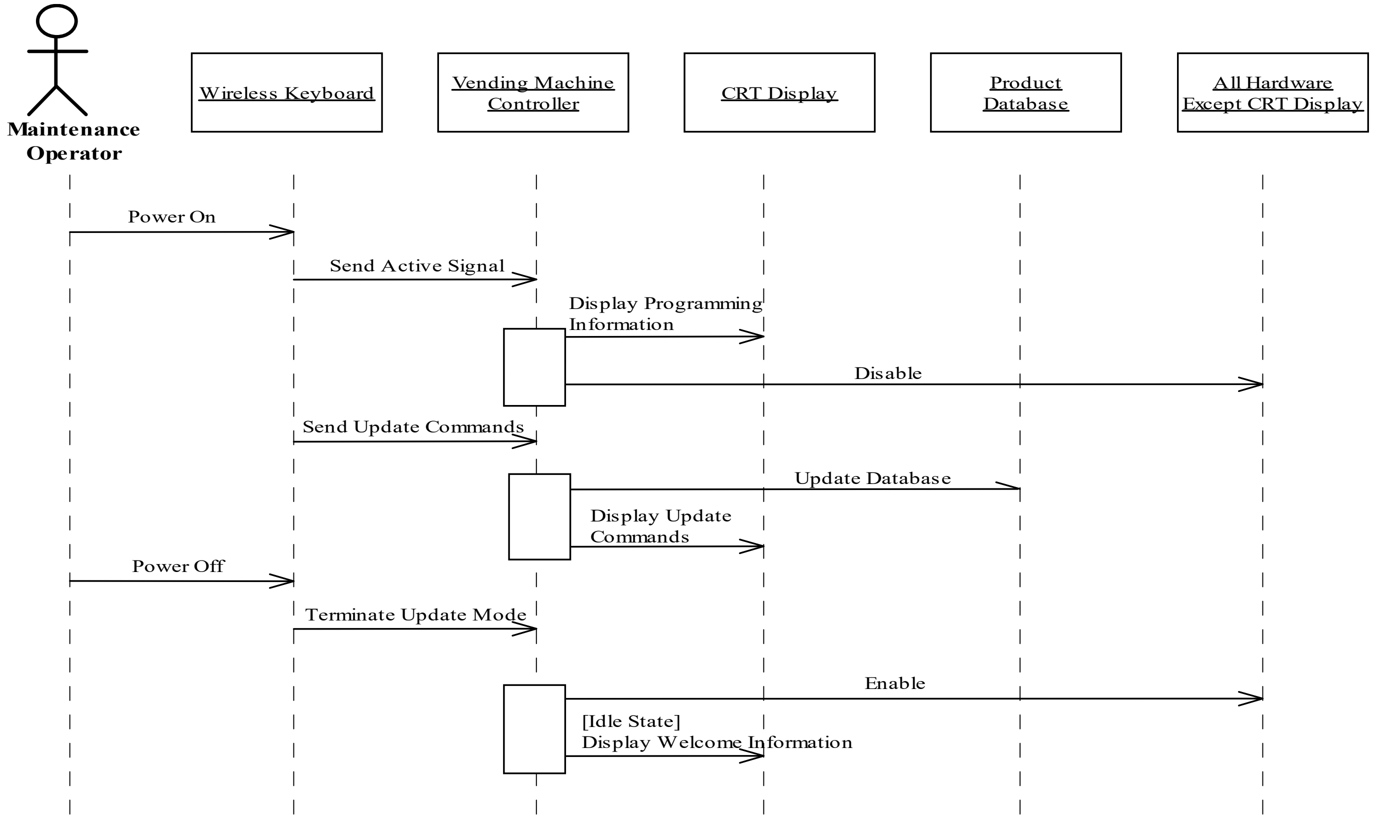
System Sequence Diagram for Make Selection Use Case


Customer
Numeric Keypad
CRT Display
Vending Machine Controller
Money Dispenser
Product Dispenser
Product Database
Note: If Release-Input Button Pressed see Cancellation Sequence Diagram
Prompt for Selection
Enter Selection
Display Selection
Send Selection
Inquire Database
Return Response
Note: Insufficient money, selection not available
Validate
[if Invalid Selection] Prompt for Another Selection
Update Database
Dispense Product
Dispense Calculated Change
[Idle State] Display Welcome Message

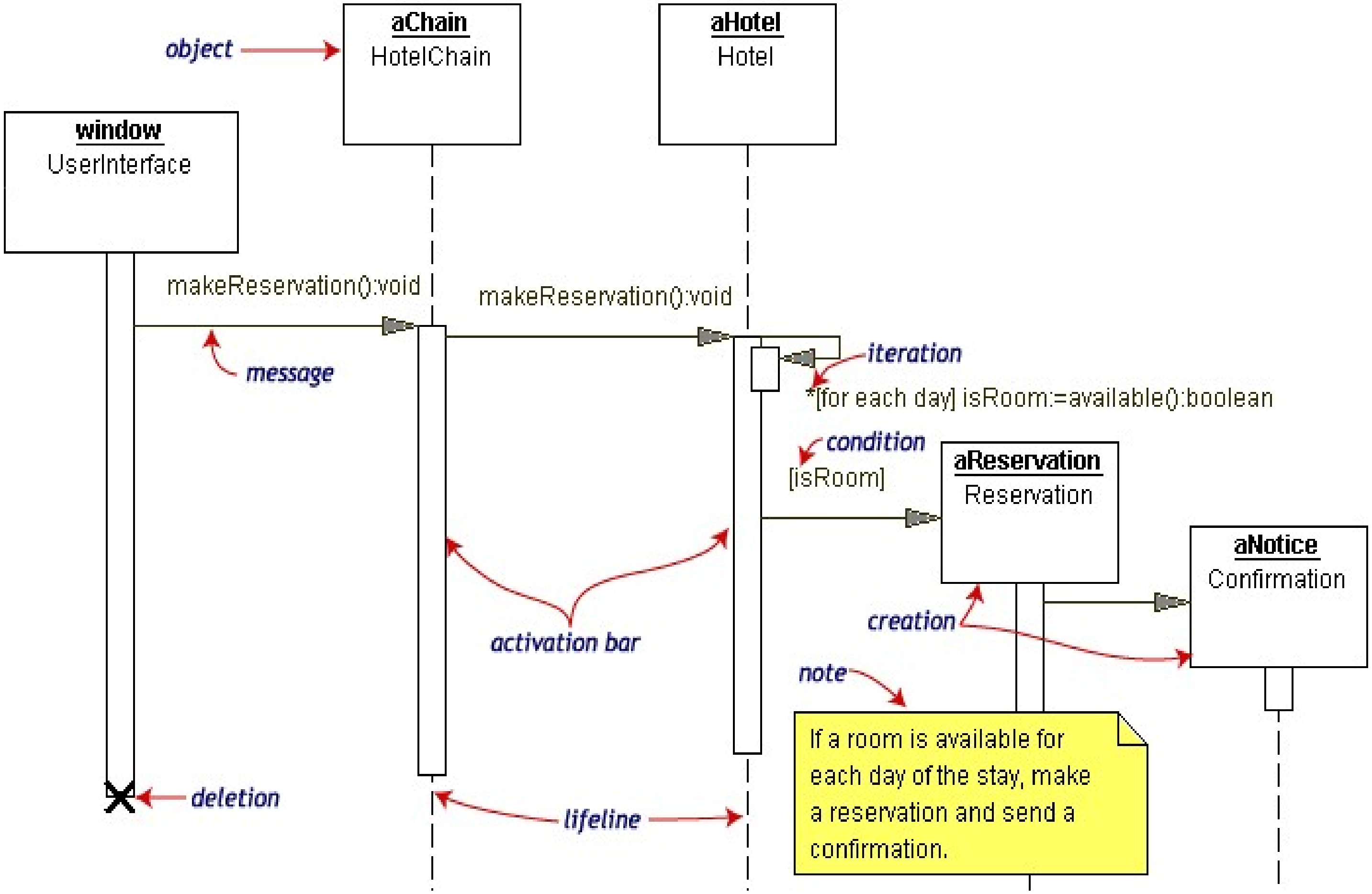
System Sequence Diagram for Cancellation Use Case


Customer
Numeric Keypad
Vending Machine Controller
CRT Display
Money Dispenser
Prompt for Selection
Release-Input Button Pressed
Send Cancellation Signal
Return Money
[Idle State] Display Welcome Message

System Sequence Diagram for Update Database Use Case

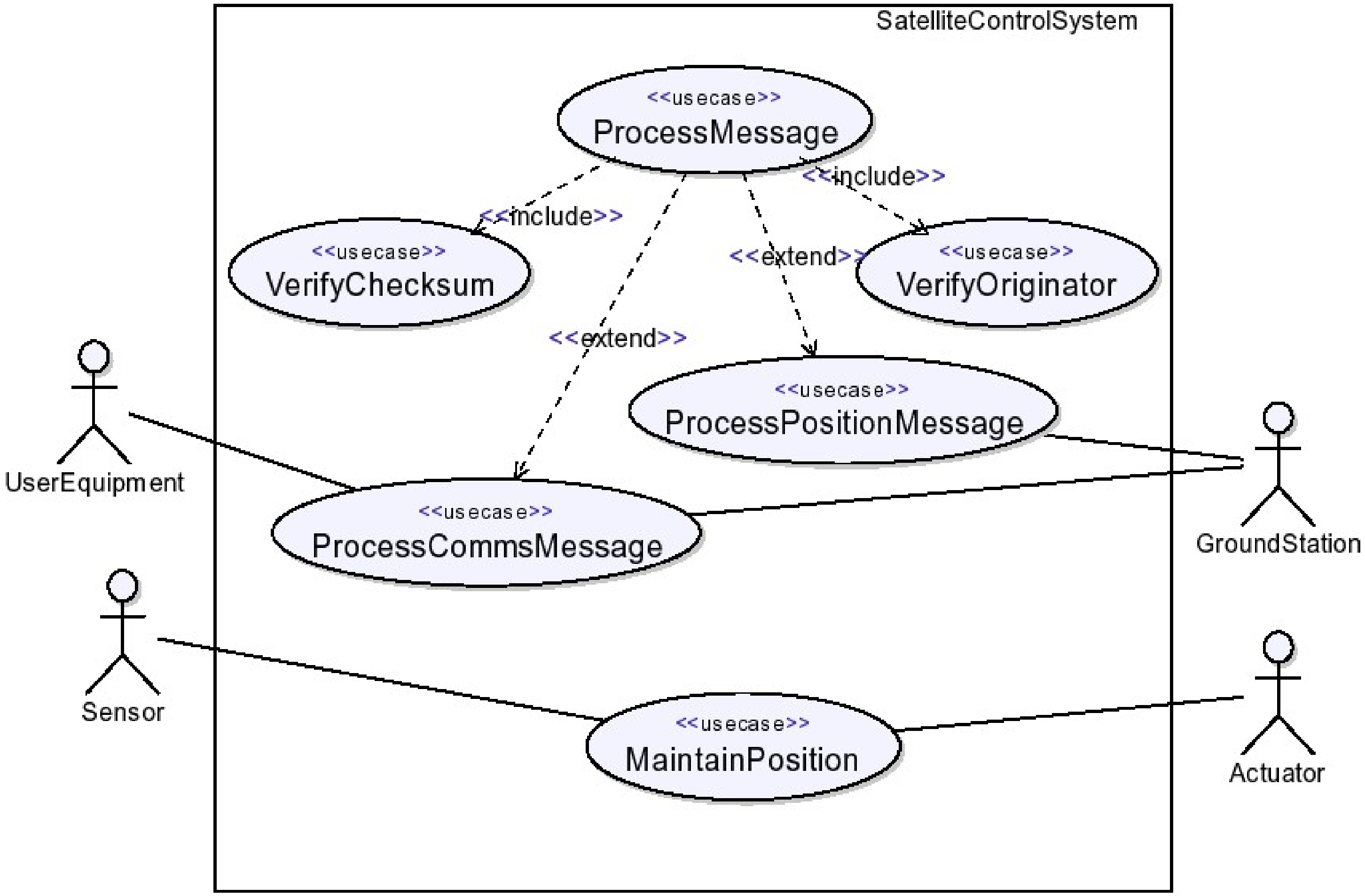

Maintenance Operator
Wireless Keyboard
Vending Machine Controller
CRT Display
Product Database
All Hardware Except CRT Display
Power On
Send Active Signal
Display Programming Information
Disable
Send Update Commands
Update Database
Display Update Commands
Power Off
Terminate Update Mode
Enable
[Idle State]
Display Welcome Information

Other Examples of Sequence Diagrams

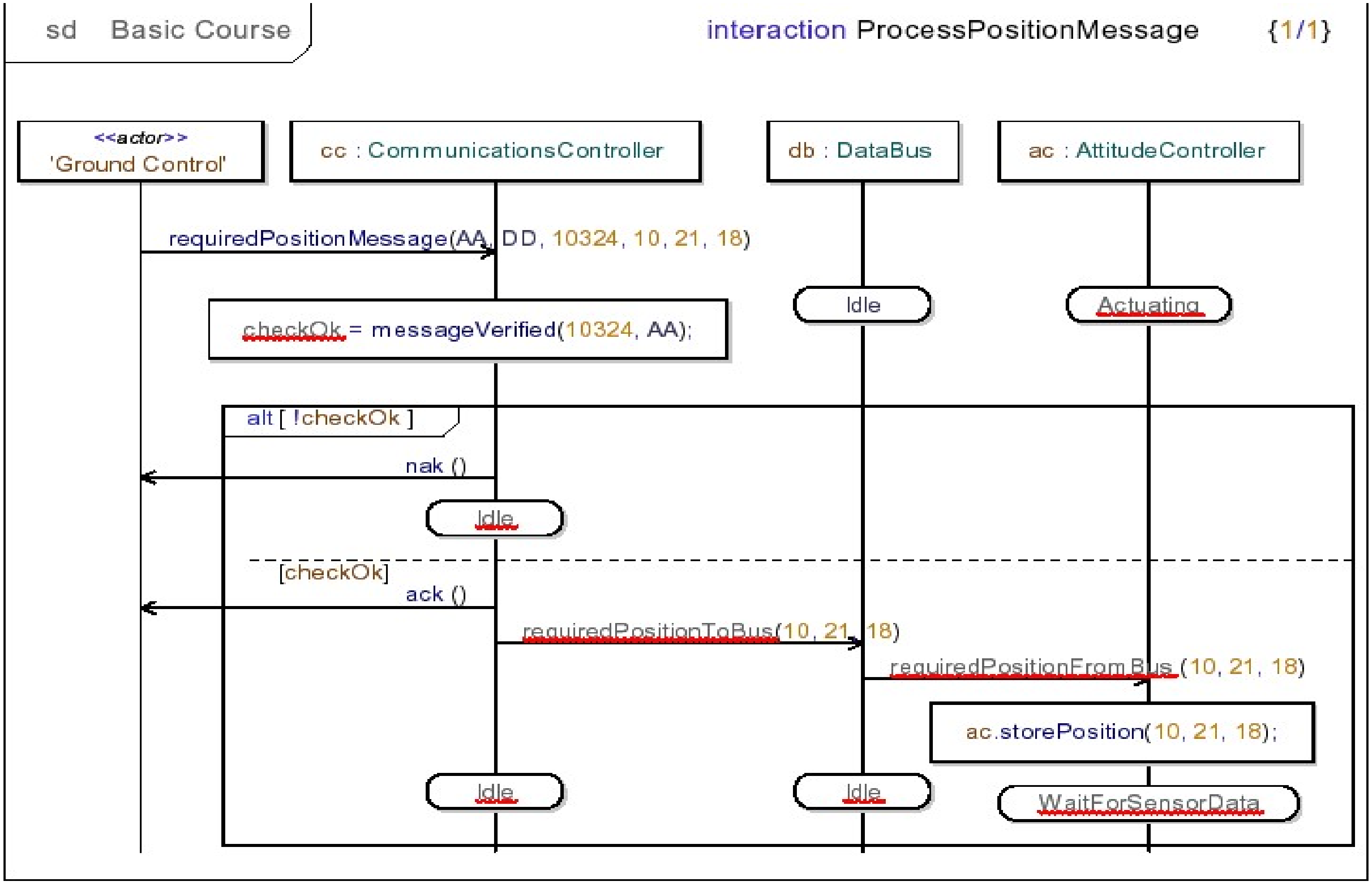

object
aChain
HotelChain
aHotel
Hotel
window
UserInterface
makeReservation():void
makeReservation():void
message
iteration
*[for each day] isRoom:=available():boolean
condition
[isRoom]
aReservation
Reservation
aNotice
Confirmation
activation bar
creation
note
deletion
lifeline

If a room is available for each day of the stay, make a reservation and send a confirmation.

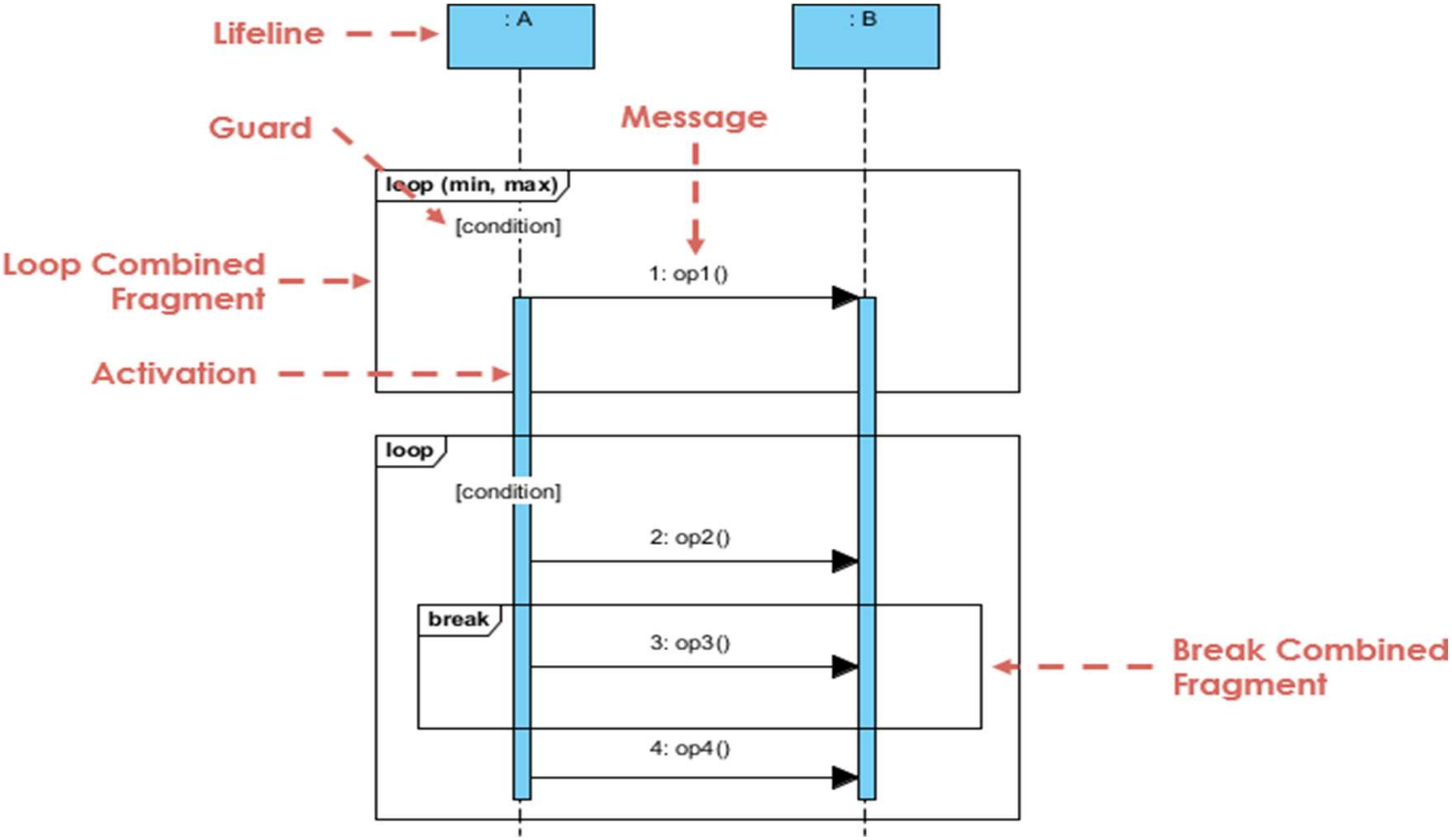
A Simple Example of Using UML 2 - EXAMPLE: SATELLITE CONTROL SYSTEM


SatelliteControlSystem
<<usecase>>
ProcessMessage
<<include>>
<<include>>
<<usecase>>
VerifyChecksum
<<extend>>
<<usecase>>
VerifyOriginator
<<extend>>
<<usecase>>
ProcessPositionMessage
UserEquipment
<<usecase>>
ProcessCommsMessage
GroundStation
Sensor
<<usecase>>
MaintainPosition
Actuator

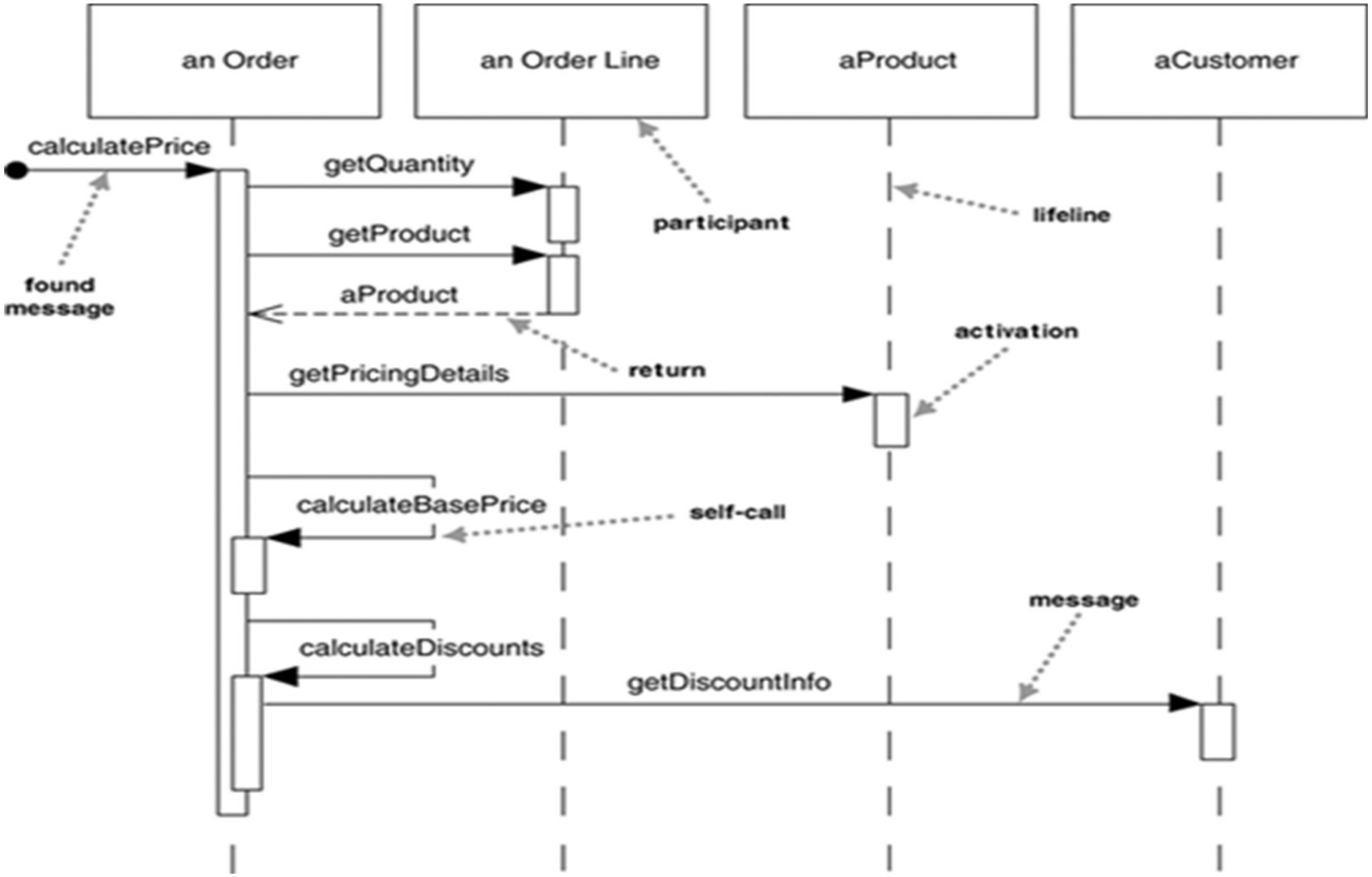
A Simple Example Using UML 2 - SATELLITE CONTROL SYSTEM Architectural behavior


sd Basic Course
interaction ProcessPositionMessage {1/1}
<<actor>>
'Ground Control'
cc : CommunicationsController
db : DataBus
ac : AttitudeController
requiredPositionMessage(AA, DD, 10324, 10, 21, 18)
Idle
Actuating
checkOk = messageVerified(10324, AA);
alt [!checkOk]
nak ()
Idle
[checkOk]
ack ()
requiredPositionToBus(10, 21, 18)
requiredPositionFromBus (10, 21, 18)
ac.storePosition(10, 21, 18);
Idle
Idle
WaitForSensorData

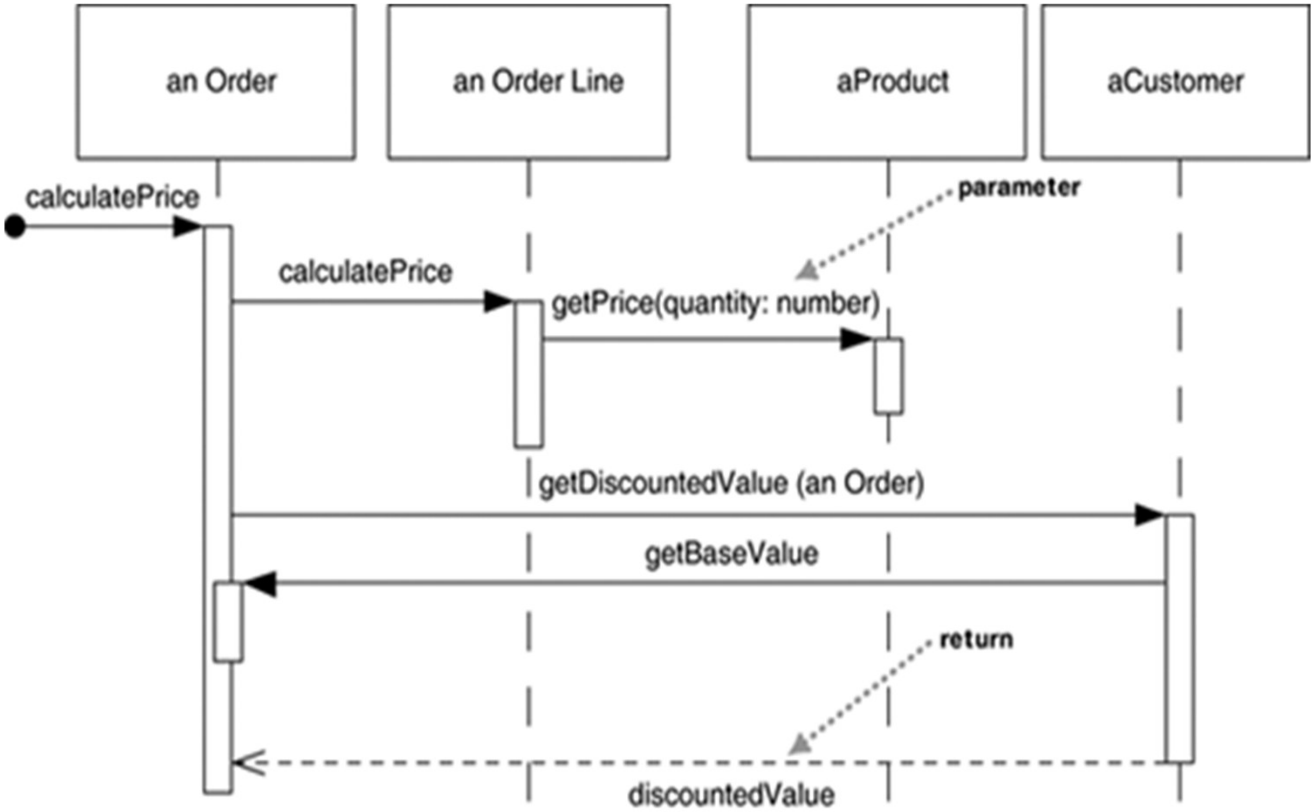
Conditional Interaction


Lifeline
: A
: B
Guard
Message
loop (min, max)
[condition]
Loop Combined Fragment
1: op1()
Activation
loop
[condition]
2: op2()
break
3: op3()
Break Combined Fragment
4: op4()

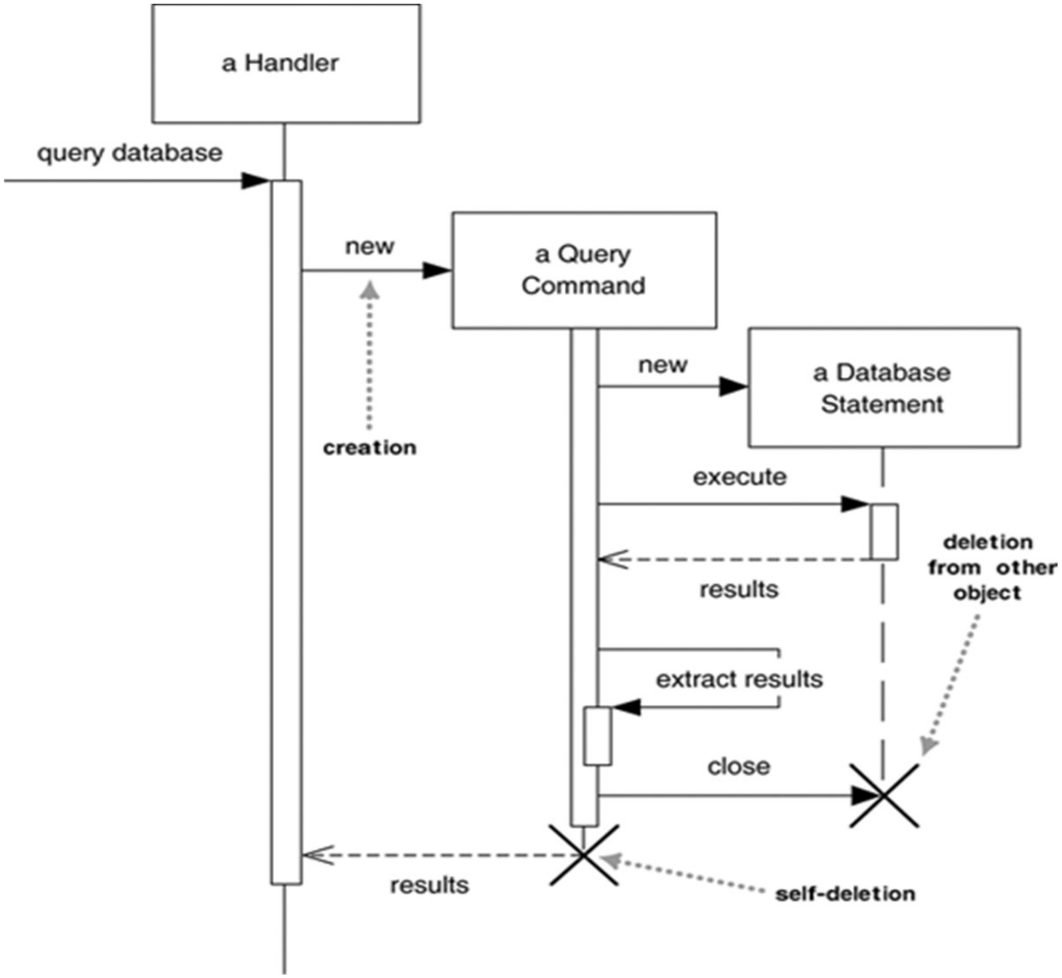
A sequence diagram for centralized control

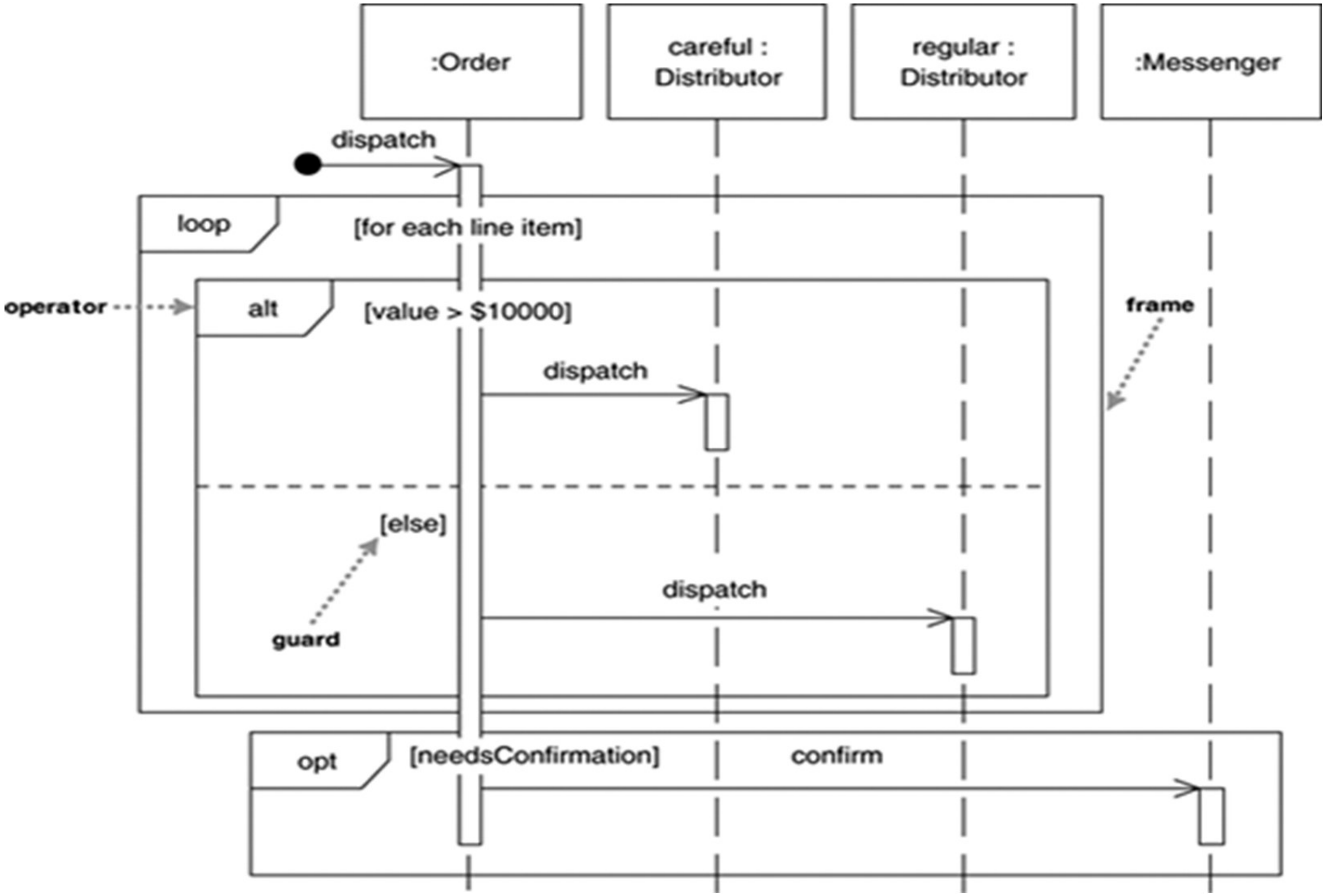

an Order
an Order Line
aProduct
aCustomer
calculatePrice
getQuantity
getProduct
aProduct
found message
participant
lifeline
activation
return
getPricingDetails
calculateBasePrice
self-call
calculateDiscounts
message
getDiscountInfo

A sequence diagram for distributed control


an Order
an Order Line
aProduct
aCustomer
calculatePrice
parameter
calculatePrice
getPrice(quantity: number)
getDiscountedValue (an Order)
getBaseValue
return
discountedValue

Creation and deletion of participants


a Handler
query database
new
a Query Command
creation
new
a Database Statement
execute
results
deletion from other object
extract results
close
results
self-deletion


:Order
careful : Distributor
regular : Distributor
:Messenger
dispatch
loop
[for each line item]
operator
alt
[value > $10000]
frame
dispatch
[else]
guard
dispatch
opt
[needsConfirmation]
confirm

Summary

- In this session we have learned
- What is meant by use case realization
- How to realise use cases with robustness analysis and communication diagram
- How the CRC technique helps identify classes and allocate responsibilities
- How to assemble the analysis class diagram
- Sequence diagram – Iteration, conditional interactions

BITS Pilani
Pilani Campus

SOFTWARE DESIGN PRINCIPLES

Overview of SOLID Principles
By Prof A R Rahman

SOLID PRINCIPLES

SOLID principles

S ⇒ stands for Single Responsibility Principle(SRP)

O ⇒ stands for Open Closed Principle(OCP)

L ⇒ stands for Liskov’s Substitution Principle(LSP)

I ⇒ stands for Interface Segregation Principle(ISP)

D ⇒ stands for Dependency Inversion Principle(DIP)

SOLID principles

- *SOLID* is the acronym for a set of practices that, when implemented together, makes the code more adaptive to change.
- Bob Martin and Micah Martin introduced these concepts in their book *‘Agile Principles, Patterns, and Practices’*.
- These five principles have changed the world of object-oriented programming, and also changed the way of writing software.
- It also ensures that the software is modular, easy to understand, debug, and refactor.
- we will discuss SOLID principles with proper example.

SOLID PRINCIPLES

- The word SOLID acronym for:
- Single Responsibility Principle (SRP)
- Open-Closed Principle (OCP)
- Liskov Substitution Principle (LSP)
- Interface Segregation Principle (ISP)
- Dependency Inversion Principle (DIP)

When we use all the principles of S.O.L.I.D in a combined manner, it becomes easier for us to develop software that can be managed easily.

The other features of using S.O.L.I.D are:

- It avoids code smells.
- Quickly refactor code.
- Can do adaptive or agile software development.

What are the benefits of using SOLID Design Principles?

- Here are some of the most important purposes of following SOLID Design Principles that help in our coding.

- To make it easier to quickly extend the system with new functionality without breaking the existing ones.

- If we don’t do implementation of a new feature in the application sensibly, It may even affect the existing functionality and cause unintentional issues.

What are the benefits of using SOLID Design Principles?

- To make the code easier to read and understand.

- Thus, spend less time figuring out what it does.

- If our code doesn’t follow the SOLID design principles, we generally take more time to understand what it does.

What are the benefits of using SOLID Design Principles?

- The SOLID principle approach ensures that our code is relatively easier to read and understand.

- To make the code more maintainable.

- Every software needs upgradation from time to time.

What are the benefits of using SOLID Design Principles?

- Therefore, we need to build applications while keeping in mind the possibility of future changes.

- It is quite effortless to refactor your code with the SOLID principles.

- If we apply the SOLID principles in the code, it will also ensure an easy debugging process as a whole.

Single Responsibility Principle:

This principle states that “*a class should have only one reason to change*” which means every class should have a single responsibility or single job or single purpose.

Take the example of developing software.

The task is divided among different members doing different things as front-end designers do design, the tester does testing and backend developer takes care of backend development part then we can say that everyone has a single job or responsibility.

Most of the time it happens that when programmers have to add features or new behavior they implement everything into the existing class which is completely wrong.

It makes their code lengthy, complex and consumes time when later something needs to be modified.

Use *layers* in your application and break God classes into smaller classes or modules.

Single Responsibility Principle

- We may come across one of the principles of object-oriented design, Separation of Concerns (SoC), that conveys a similar idea. The name of the SRP says it all:

- “One class should have one and only one responsibility”

- we should write, change, and maintain a class only for one purpose.

Single Responsibility Principle

- A class is like a container.
- We can add any amount of data, fields, and methods into it.
- However, if we try to achieve too much through a single class, soon that class will become bulky.
- If we follow SRP, the classes will become compact and neat where each class is responsible for a single problem, task, or concern.

For example:

- If a given class is a model class then it should strictly represent only one actor/entity in the application.

- This kind of design decision will give us the flexibility to make changes in the class, in future without worrying the impacts of changes in other classes.

- If we are writing service/manager class then the class should contain only that part of methods and nothing else.

For example:

- The service class should not contain even utility global functions related to the module.

- Better to separate the global functions in another globally accessible class.

- This will help in maintaining the class for that particular purpose, and we can decide the visibility of class to a specific module only.

Example

- In Log4j2, we have different classes with logging methods, different classes are logging levels and so on.

- In our application level code, we define model classes to represent real time entities such as Person, Employee, Account etc.

- Most of these classes are examples of SRP principle because when we need to change the state of a person, only then we will modify the Person class, and so on.

Example

- we have two classes Person and Account.

- Both have single responsibility to store their specific information.

- If we want to change the state of Person then we do not need to modify the class Account and vice-versa.

Person.java

```
public class Person
{

  private Long personId;
  private String firstName;
  private String lastName;
  private String age;
  private List<Account> accounts;

}
```

Account.java

```
public class Account
{

  private Long guid;
  private String accountNumber;
  private String accountName;
  private String status;
  private String type;

}
```

Example

```
class Employee
 {

public Pay calculatePay() {...}
public void save() {...}
public String describeEmployee() {...}

}
```

- Here we have pay calculation logic with database logic and reporting logic all mixed up within one class.
- If you have multiple responsibilities combined into one class, it might be difficult to change one part without breaking others.
- Mixing responsibilities also makes the class harder to understand and harder to test, decreasing cohesion.
- The easiest way to fix this is to split the class into three different classes, with each having only one responsibility: database access, calculating pay, and reporting, all separated.

Single Responsibility Principle

- There should never be more than one reason for a class to change.
- A class should be focused on a single functionality, address a specific concern.
- This means that every class, or similar structure, in your code should have only one job to do.
- Everything in the class should be related to that single purpose, i.e., be cohesive.
- It does not mean that our classes

Single Responsibility Principle

- The more responsibilities our class has, the more often we need to change it.
- If our class implements multiple responsibilities, they are no longer independent of each other.
- Therefore, we will need to change our class more often.
- Consequently, it will have more side-effects, and requires a lot more work than it should have.
- So, it’s better to avoid these problems by making sure that each class has only one responsibility.
- should only contain one method or property.

Single Responsibility Principle

- Suppose we built our software over a longer period and we need to adapt it to ongoing change requests.

- we might feel the easiest and fastest approach is adding a method or functionality to our existing code instead of writing a new class or component.

- But that often results in classes with more than responsibility and makes it more and more difficult to maintain the software.

- we can avoid these problems by asking a simple question before we make any changes:

- *What is the responsibility of our class/component/microservice?*

- *If our answer includes the word "and", you're most likely breaking the Single Responsibility Principle*.

- Then it's better to take a step back and rethink your current approach.

Single Responsibility Principle

How many responsibilities? we have 1) calculation logic 2) database logic and 3) reporting logic.

```
public class Employee {
   public Double calculatePay() {...}
   public void saveEmployee() {...}
   public void getEmployeeReport() {...}
}
```

- All mixed up within one class.
- If we have multiple responsibilities combined into one class, it might be difficult to change one part without breaking others.
- Mixing responsibilities also makes the class harder to understand and harder to test.
- It will also decrease the cohesion.
- The easiest way to fix this is to split the class into three different classes, with each having only one responsibility: database access, calculating payment and reporting, all separated.

Single Responsibility Principle

There are 3 reasons for this class to change.

- 1) Any change in payment calculation.

- 2) Any change in logic of saving employee into the database.

- 3) Any change in reporting logic.

Single Responsibility Principle

- If we think in a broad way, in order to make changes in one part, we may also have to change all parts unnecessarily.

- We will split the Employee class into three classes in order to follow the Single Responsibility Principle.

- For example, we change the code to demonstrates the concept.

```
public class EmployeePayment {

      public Double calculatePay() {

  // logic to calculate employee payment

       }
}
```

```
public class EmployeeReport {

        public void getEmployeeReport() {

         // logic to generate employee report

            }
}
```

```
public class EmployeeRepository {

           public void saveEmployee() {

     // logic to save employee object

       }
 }
```

- Now we can call these methods from the respective classes wherever it is required.
- Don’t get confused with the fact that a class will have only one method to follow SRP.
- It can have many supporting methods, but they should not change the responsibility of the class.

Open/Closed Principle

Open/Closed Principle:

- This principle states that “*software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification*” which means you should be able to extend a class behavior, without modifying it.

- Suppose developer A needs to release an update for a library or framework and developer B wants some modification or add some feature on that then developer B is allowed to extend the existing class created by developer A but developer B is not supposed to modify the class directly.

- Using this principle separates the existing code from the modified code so it provides better stability, maintainability and minimizes changes as in your code.

Open/Closed Principle:

- OCP is the second principle which we should keep in mind while designing our application. It states:

- “Software components should be open for extension, but closed for modification”

- It means that the application classes should be designed in such a way that whenever fellow developers want to change the flow of control in specific conditions in application, all they need to extend the class and override some functions and that’s it.

- If other developers are not able to write the desired behavior due to constraints put by the class, then we should reconsider refactoring the class.

- we do not mean here that anybody can change the whole logic of the class, but one should be able to override the options provided by software in a non harmful way permitted by the software.

Example

- If we take a look into any good framework like struts or spring.

- we will see that we can not change their core logic and request processing, but we modify the desired application flow just by extending some classes and plugin them in configuration files.

- Spring framework has class DispatcherServlet.

- This class acts as a front controller for String based web applications.

Example

- To use this class, we are not required to modify this class.

- All we need is to pass initialization parameters and we can extend its functionality the way we want.

- Please note that apart from passing initialization parameters during application startup, we can override methods as well to modify the behavior of target class by extending the classes.

- For example, struts Action classes are extended to override the request processing logic.

Extending Struts Action

```
public class HelloWorldAction extends Action   {

        @Override

         public ActionForward execute(ActionMapping mapping, ActionForm form,
                  HttpServletRequest request,
                  HttpServletResponse response)

          throws Exception
 {

     //Process the request

   }
}
```

Open/Closed Principle:

- The Open-Closed Principle (OCP) states that classes should be open for extension but closed for modification.

- “Open to extension” means that you should design your classes so that new functionality can be added as new requirements are generated.

- “Closed for modification” means that once we have developed a class we should never modify it, except to correct bugs.

- These two parts of the principle appear to be contradictory.

- However, if we correctly structure classes and their dependencies, we can add functionality without editing existing source code.

Open/Closed Principle:

- Generally you achieve this by referring to abstractions for dependencies, such as interfaces or abstract classes, rather than using concrete classes.
- Functionality can be added by creating new classes that implement the interfaces.
- Applying OCP to our projects limits the need to change source code once it has been written, tested, and debugged.
- This reduces the risk of introducing new bugs to existing code, leading to more robust software.
- Another side effect of the use of interfaces for dependencies is reduced coupling (i.e. when changing code in class A forces related changes in class B), and increased flexibility.

Open/Closed Principle:

```
void checkOut(Receipt receipt) {
     Money total = Money.zero;
     for (item : items) {
     total += item.getPrice();
     receipt.addItem(item);
 }
   Payment p = acceptCash(total);
    receipt.addPayment(p);
}
```

```
Payment p;
 if (credit)
     p = acceptCredit(total);
     else
      p = acceptCash(total);
receipt.addPayment(p);
```

So how do we add credit card support?
You could add an “if” statement like this, but then that would be violation of OCP.

Open/Closed Principle: Here is a better solution:

```
public interface PaymentMethod { void acceptPayment(Money total); }

void checkOut(Receipt receipt, PaymentMethod pm)  {

        Money total = Money.zero;

      for (item : items)  {
      total += item.getPrice();
       receipt.addItem(item);
      }

      Payment p = pm.acceptPayment(total);

      receipt.addPayment(p);
}
```

Open/Closed Principle:

- Keep in mind – OCP helps only if the changes that are going to come are predictable.
- so you should apply it only if a similar change has already happened.
- So, first do the simplest thing and then see what changes are requested so we can more accurately predict the future changes.
- This means waiting for a customer to make a change and then invent the abstractions that will protect yourself from a similar change in the future.

What is the benefits of Open Closed Principle?

- Some of the benefits of applying open closed principle in your code :

- Easier Extensibility

- Needless to say, Open-closed Principle offers a better extensibility to your code.

- When you write your code in such a way that it has no modification and allows extension, you will get better extensibility on it’s own.

What is the benefits of Open Closed Principle?

- Easier to Maintain
- The Principle suggests using interfaces.
- Interfaces in code offers an additional level of abstraction which in turn enables loose coupling.
- The implementations of an interface are independent of each other and don’t need to share any code.
- Hence, we can easily maintain our code with client’s keep changing requirements.

What is the benefits of Open Closed Principle?

- Flexibility
- When we follow the open closed principle in writing our code, we will have better flexibility to extend it.

- Further, if any change request arises in future, your code will be more flexible to extend.

- limitations of Open Closed Principle
- Whenever we incorporate any change to the code as an extension, we need to do unit testing before and after implementing the same.

write a program that calculates area of various shapes

```
public class Rectangle {

        public Double length;

         public Double width;

 }



  public class Circle {

      public Double radius;

   }
```

```
public class AreaCalculator {

       public Double calculateRectangleArea(Rectangle rectangle){

                 return rectangle.length * rectangle.width;

        }

}
```

There are flaws in the way of designing our solution

```
public class AreaCalculator{


    public Double calculateRectangleArea(Rectangle rectangle){

       return rectangle.length * rectangle.width;

    }
       public Double calculateCircleArea(Circle circle){


            return (22 / 7) * circle.radius * circle.radius;
      }

}
```

Code that follows OCP

```
public interface Shape {
 public Double calculateArea();

}
 public class Rectangle implements Shape {
            Double length;
            Double width;
    public double calculateArea() {

        return length * width;

     }
}

public class Circle implements Shape {

   public Double radius;

   public Double calculateArea() {

         return (22 / 7) * radius * radius;

   }
}
```

```
public class AreaCalculator {

   public Double calculateShapeArea(Shape shape) {

                 return shape.calculateArea();

    }
}
```

Liskov’s Substitution Principle

Liskov’s Substitution Principle:

- The principle was introduced by Barbara Liskov in 1987 and according to this principle

- “*Derived or child classes must be substitutable for their base or parent classes*“.

- This principle ensures that any class that is the child of a parent class should be usable in place of its parent without any unexpected behavior.

Liskov’s Substitution Principle:

- we can understand it in a way that a farmer’s son should inherit farming skills from his father and should be able to replace his father if needed.

- If the son wants to become a farmer then he can replace his father but if he wants to become a cricketer then definitely the son can’t replace his father even though they both belong to the same family hierarchy.

- if class *A* is a subtype of class *B*, then we should be able to replace objects of *B* with objects of *A* (i.e., objects of type A may substitute objects of type B) without changing the behavior (correctness, functionality, etc.) of our program.

Liskov's Substitution Principle:

- One of the classic examples of this principle is a rectangle having four sides.
- A rectangle's height can be any value and width can be any value.
- A square is a rectangle with equal width and height.
- So we can say that we can extend the properties of the rectangle class into square class.
- In order to do that we need to swap the child (square) class with parent (rectangle) class to fit the definition of a square having four equal sides but a derived class does not affect the behavior of the parent class, so if we do that it will violate the Liskov Substitution Principle.

Liskov's Substitution Principle:

- LSP is a variation of previously discussed open closed principle. It says:
- "Derived types must be completely substitutable for their base types"
- LSP means that the classes, fellow developers created by extending our class, should be able to fit in application without failure.

Liskov's Substitution Principle:

- This is important when we resort to polymorphic behavior through inheritance.
- This requires the objects of the subclasses to behave in the same way as the objects of the superclass.
- This is mostly seen in places where we do runtime type identification and then cast it to appropriate reference type.

Why is it required to follow Liskov Substitution Principle?

- This avoids overuse/misuse of inheritance.
- It helps us conform to the “is-a” relationship.
- We can also say that subclasses must fulfil a contract defined by the base class.

Example

- An example of LSP can be custom property editors in Spring framework.

- Spring provides property editors to represent properties in a different way than the object itself

- e.g. parsing human readable inputs from HTTP request parameters or displaying human readable values of pure java objects in view layer e.g. Currency or URL.

Example

- Spring can register one property editor for one data type and it is required to follow the constraint mandated by base class PropertyEditorSupport.
- So if any class extends PropertyEditorSupport class, then it can be substituted by everywhere the base class is required.
- For example, every book has an ISBN number which is always be in a fixed display format.
- You can have separate representations of ISBN in database and UI.
- For this requirement, we may write property editor in such a way –

IsbnEditor.java

```
import java.beans.PropertyEditorSupport;

import org.springframework.util.StringUtils;

import com.howtodoinjava.app.model.Isbn;

public class IsbnEditor extends PropertyEditorSupport {

   @Override
   public void setAsText(String text) throws IllegalArgumentException {

      if (StringUtils.hasText(text)) {

         setValue(new Isbn(text.trim()));

      }
        else {

         setValue(null);

      }
 }
```

```
@Override

   public String getAsText() {

      Isbn isbn = (Isbn) getValue();

      if (isbn != null) {

         return isbn.getIsbn();

      }
      else {

         return "";

      }
   }
}
```

Liskov's Substitution Principle:

- The Liskov Substitution Principle (LSP) applies to inheritance hierarchies, specifying that we should design our classes, so that client dependencies can be substituted with subclasses without the client knowing about the change.

- All subclasses must, therefore, operate in the same manner as their base classes.

- The specific functionality of the subclass may be different but must conform to the expected behavior of the base class.

- To be a true behavioral subtype, the subclass must not only implement the base class's methods and properties, but also conform to its implied behavior.

Liskov's Substitution Principle:

- In general, if a subtype of the supertype does something that the client of the supertype does not expect, then this is in violation of LSP.

- Imagine a derived class throwing an exception that the superclass does not throw, or if a derived class has some unexpected side effects.

- Basically, derived classes should never do less than their base class.

Liskov's Substitution Principle:

- A typical example that violates LSP is a Square class that derives from a Rectangle class.
- The Square class always assumes that the width is equal with the height.
- If a Square object is used in a context where a Rectangle is expected, unexpected behavior may occur because the dimensions of a Square cannot (or rather should not) be modified independently.

Liskov's Substitution Principle:

- This problem cannot be easily fixed:
- if we can modify the setter methods in the Square class so that they preserve the Square invariant (i.e., keep the dimensions equal), then these methods will weaken (violate) the post-conditions for the Rectangle setters, which state that dimensions can be modified independently.

```
public class Rectangle {

    private double height;
    private double width;

public double area();

public void setHeight(double height);
public void setWidth(double width);


}
```

```
public class Square extends Rectangle {

   public void setHeight(double height) {

         super.setHeight(height);
         super.setWidth(height);
   }
  public void setWidth(double width) {
          setHeight(width);
   }
}
```

What you see above violates LSP.

Violations of LSP cause undefined behavior, which can lead to weeks of wasted time trying to find out where the bug is.

create a BookDelivery class that informs customers about the number of locations

```
public class BookDelivery {

  String titles;     Integer userID;

   void getDeliveryLocations() {

           ...

     }
}
public class PosterMapDelivery extends BookDelivery {

        @Override

    void getDeliveryLocations() {

                  ...

     }
}
```

```
public class AudioBookDelivery extends BookDelivery {

        @Override
        void getDeliveryLocations() {

                 /* can't be implemented since

                  * Audio book doesn't have

                  * a physical location. */

         }
}
```

Solution

- In order to solve the problem, we need to fix the inheritance hierarchy.
- Let’s introduce an extra layer that better differentiates book delivery types.
- The new OfflineDelivery and OnlineDelivery classes split up the BookDelivery superclass.
- We will also move the getDeliveryLocations() method to OfflineDelivery.
- Next, we will create a new getSoftwareOptions() method for the OnlineDelivery class (as this is more suitable for online deliveries).

Corrected code

- PosterMapDelivery will be the child class of OfflineDelivery and it will override the getDeliveryLocations() method with its own functionality.

- AudioBookDelivery will be the child class of OnlineDelivery which is good news, as now it doesn’t have to deal with the getDeliveryLocations() method.

- Instead, it can override the getSoftwareOptions() method of its parent with its own implementation (for instance, by listing and embedding available audio players).

Corrected code

- we could use any subclass in place of its superclass without breaking the application.

- Violations of LSP cause undefined behavior.

- Undefined behavior means that it works okay during development but blows up in production, or that you spend weeks debugging something that only occurs once per day, or that you have to go through hundreds of megabytes of logs to figure out what went wrong.

```
public class BookDelivery {

          String title;
          Integer userID;

 }
 public class OfflineDelivery extends BookDelivery {

           void getDeliveryLocations() {
                       ...
          }

 }
public class OnlineDelivery extends BookDelivery {


           void getSoftwareOptions() {
                            ...
          }
}
```

PosterMapDelivery will be the child class of OfflineDelivery and it will override the getDeliveryLocations() method with its own functionality.

AudioBookDelivery will be the child class of OnlineDelivery which is good news, as now it doesn’t have to deal with the getDeliveryLocations() method.

Instead, it can override the getSoftwareOptions() method of its parent with its own implementation (for instance, by listing and embedding available audio players).

```
public class PosterMapDelivery extends OfflineDelivery {

        @Override
        void getDeliveryLocations() {
            ...

        }
}

public class AudioBookDelivery extends OnlineDelivery {

        @Override
        void getSoftwareOptions() {


                ...
        }
}
```

Benefit of Liskov Substitution Principle

- If our code follows the Liskov’s Substitutional Principle, we will have below benefits:
- 1) Code Reusability
- 2) Easier Maintenance
- 3) Reduced Coupling

Interface Segregation Principle

Interface Segregation Principle:

This principle is the first principle that applies to Interfaces instead of classes in SOLID and it is similar to the single responsibility principle.

It states that “*do not force any client to implement an interface which is irrelevant to them*“.

Here our main goal is to focus on avoiding fat interface and give preference to many small client-specific interfaces.

We should prefer many client interfaces rather than one general interface and each interface should have a specific responsibility.

Interface Segregation Principle:

Suppose if we enter a restaurant and we are pure vegetarian.

The waiter in that restaurant gave us the menu card which includes vegetarian items, non-vegetarian items, drinks, and sweets.

In this case, as a customer, we should have a menu card which includes only vegetarian items, not everything which we don’t eat in our food.

Here the menu should be different for different types of customers.

The common or general menu card for everyone can be divided into multiple cards instead of just one.

Using this principle helps in reducing the side effects and frequency of required changes.

Interface Segregation Principle:

- This principle ISP is applicable to interfaces as a single responsibility principle holds to classes. ISP says:
- “Clients should not be forced to implement unnecessary methods which they will not use”
- Let us take an example. A Developer created an interface Reportable and added two methods generateExcel() and generatedPdf().
- Now client ‘A’ wants to use this interface but he intends to use reports only in PDF format and not in excel.
- Will he/She be able to use the functionality easily?

Interface Segregation Principle:

- NO. He/She will have to implement both the methods, out of which one is an extra burden put on him by the designer of the software.
- Either he/She will implement another method or leave it blank.
- This is not a good design.
- So what is the solution? Solution is to create two interfaces by breaking the existing one.
- They should be like PdfReportable and ExcelReportable.
- This will give the flexibility to users to use only the required functionality only.

Example

- The best place to look for ISP examples is Java AWT event handlers for handling GUI events fired from keyboard and mouse.

- It has different listener classes for each kind of event.

- We only need to write handlers for events, we wish to handle.

- Nothing is mandatory.

Example

- Some of the listeners are –
- FocusListener
- KeyListener
- MouseMotionListener
- MouseWheelListener
- TextListener
- WindowFocusListener
- Anytime, we wish to handle any event, just find out a corresponding listener and implement it.

MouseMotionListenerImpl.java

```
public class MouseMotionListenerImpl implements MouseMotionListener {

        @Override
        public void mouseDragged(MouseEvent e) {

            //handler code

        }

        @Override
        public void mouseMoved(MouseEvent e) {

           //handler code
        }
}
```

- The Interface Segregation Principle (ISP) states that clients should not be forced to depend upon interface members they do not use.
- When we have non-cohesive interfaces, the ISP guides us to create multiple, smaller, cohesive interfaces.
- When we apply ISP, classes and their dependencies communicate using tightly-focused interfaces, minimizing dependencies on unused members and reducing coupling accordingly.
- Smaller interfaces are easier to implement, improving flexibility and the possibility of reuse.
- As fewer classes share these interfaces, the number of changes that are required in response to an interface modification is lowered, which increases robustness.

- Let us consider our ATM, which has a screen where we wish to display different messages.

- If we want to add a message on the ATM that appears only for withdrawal functionality, how would we solve it?

- Perhaps we would add a method to the Messenger interface and be done with it.

- But this causes we need to recompile all the users of this interface and almost all the system needs to be redeployed, which is in direct violation of OCP.

- What happened here was that changing the withdrawal functionality caused changes to other totally unrelated functionalities as well, which is something we now know we don't want.

- How did this happen?

- Actually, there is backwards dependency at play, where each functionality that uses this Messengers interface depends on methods, it does not need but are needed by other functionalities.

- Here is what we want to avoid:

```
public interface Messenger {

 askForCard();
 tellInvalidCard();
 askForPin();
 tellInvalidPin();
 tellCardWasSiezed();
 askForAccount();

tellNotEnoughMoneyInAccount();
 tellAmountDeposited();
 tellBalance();

 }
```

```
public interface LoginMessenger {

     askForCard();
     tellInvalidCard();
     askForPin();
     tellInvalidPin();

 }

public interface WithdrawalMessenger {
         tellNotEnoughMoneyInAccount();
         askForFeeConfirmation();
}

public class EnglishMessenger implements LoginMessenger,
WithdrawalMessenger {
...
 }
```

Instead, split the Messenger interface up so that different ATM functionality depend on separate Messengers.

Similarity between Interface Segregation Principle and Single Responsibility Principle

- Both the Interface Segregation Principle and Single Responsibility Principle have approximately the same goal: ensuring small, focused, and highly cohesive software components.

- The difference is that the Single Responsibility Principle is concerned with classes, while the Interface Segregation Principle is concerned with interfaces.

- Interface Segregation Principle is easy to understand and simple to follow.

Similarity between Interface Segregation Principle and Single Responsibility Principle

- But, identifying the distinct interfaces can sometimes be a challenge.

- Hence, we require careful considerations to avoid the expansion of interfaces.

- Therefore, while writing an interface, consider the possibility of implementation classes having different sets of behaviors.

- And if so, segregate the interface into multiple interfaces, each having a specific role.

Benefit of Interface Segregation Principle

- Increased code readability
- Easier to Implement
- Easier to Maintain
- Better organization of code
- Don’t need to throw exceptions unnecessarily

Dependency injection

Dependency injection

- Every dependency in the design should target an interface or an abstract class. No dependency should target a concrete class.

- In software engineering, dependency injection is a design pattern in which an object receives other objects that it depends on.

- A form of inversion of control, dependency injection aims to separate the concerns of constructing objects and using them, leading to loosely coupled programs.

- The pattern ensures that an object which wants to use a given service should not have to know how to construct those services.

- Instead, the receiving object (or 'client') is provided with its dependencies by external code (an 'injector'), which it is not aware of.

Dependency injection solves the following problems:

- How can a class be independent from the creation of the objects it depends on?

- How can an application, and the objects it uses support different configurations?

- How can the behavior of a piece of code be changed without editing it directly?

Dependency injection solves the following problems:

- Fundamentally, dependency injection consists of passing parameters to a method.

- Because the client does not build or find the service itself, it typically only needs to declare the interfaces of the services it uses, rather than their concrete implementations.

- This makes it easier to change which services are actually used at runtime, especially in statically-typed languages where changing the underlying objects would otherwise require re-compiling the source code.

- An example of inversion of control without dependency injection is the template method pattern, where polymorphism is achieved through subclassing.

- In contrast, dependency injection implements inversion of control through composition, and is often similar to the strategy pattern.

- A difference is that the strategy pattern is intended for dependencies that are interchangeable throughout an object's lifetime, whereas with dependency injection typically only a single instance of a dependency is used.

Advantages

- A basic benefit of dependency injection is decreased coupling between classes and their dependencies.

- By removing a client's knowledge of how its dependencies are implemented, programs become more reusable, testable and maintainable.

- This also results in increased flexibility: a client may act on anything that supports the intrinsic interface the client expects.

- More generally, dependency injection reduces boilerplate code, since all dependency creation is handled by a singular component.

Advantages

- Finally, dependency injection allows concurrent development.

- Two developers can independently develop classes that use each other, while only needing to know the interface the classes will communicate through.

- Plugins are often developed by third-parties that never even talk to developers of the original product.

Advantages

- Testing

- Many of dependency injection's benefits are particularly relevant to unit-testing.

- For example, dependency injection can be used to externalize a system's configuration details into configuration files, allowing the system to be reconfigured without recompilation.

- Separate configurations can be written for different situations that require different implementations of components.

- Similarly, because dependency injection does not require any change in code behavior, it can be applied to legacy code as a refactoring.

- This makes clients more independent and are easier to unit test in isolation, using stubs or mock objects, that simulate other objects not under test.

- This ease of testing is often the first benefit noticed when using dependency injection.

Disadvantages

- Critics of dependency injection argue that it:
- Creates clients that demand configuration details, which can be onerous when obvious defaults are available.
- Makes code difficult to trace because it separates behavior from construction.
- Is typically implemented with reflection or dynamic programming, hindering IDE automation.
- Typically requires more upfront development effort.
- Forces complexity out of classes and into the links between classes which might be harder to manage.
- Encourages dependence on a framework.

Dependency Inversion Principle

Dependency Inversion Principle:

Before we discuss this topic keep in mind that Dependency Inversion and Dependency Injection both are different concepts.

Most of the people get confused about it and consider both are the same.

Now two key points are here to keep in mind about this principle

- High-level modules/classes should not depend on low-level modules/classes.
- Both should depend upon abstractions.
- Abstractions should not depend upon details.
- Details should depend upon abstractions.
- The above lines simply state that if a high module or class will be dependent more on low-level modules or class then your code would have tight coupling and if you will try to make a change in one class it can break another class which is risky at the production level.

- The Dependency Inversion Principle (DIP) states that high-level modules should not depend upon low-level modules; they should depend on abstractions.
- Secondly, abstractions should not depend upon details; details should depend upon abstractions.
- This way, instead of high-level modules depending on low-level modules, both will depend on abstractions.
- The idea is that we isolate our class behind a boundary formed by the abstractions it depends on.
- If all the details behind those abstractions change, then our class is still safe.
- This helps keep coupling low and makes our design easier to change. DIP also offers us to test things in isolation, details like database are plugins to our system.

Dependency Inversion Principle:

- So always try to make classes loosely coupled as much as we can and we can achieve this through *abstraction*.
- The main motive of this principle is decoupling the dependencies so if class A changes the class B doesn’t need to care or know about the changes.
- we can consider the real-life example of a TV remote battery.
- our remote needs a battery but it’s not dependent on the battery brand.
- we can use any XYZ brand that you want and it will work.
- So we can say that the TV remote is loosely coupled with the brand name.
- Dependency Inversion makes your code more reusable.

Dependency Inversion Principle:

- Most of us are already familiar with the words used in principle’s name. DI principle says:
- “Depend on abstractions, not on concretions”
- In other words. we should design our software in such a way that various modules can be separated from each other using an abstract layer to bind them together.

Example

The classical use of this principle of bean configuration in Spring framework.

In the spring framework, all modules are provided as separate components which can work together by simply injecting dependencies in other modules.

This dependency is managed externally in XML files.

Dependency Inversion Principle:

- These separate components are so well closed in their boundaries that we can use them in other software modules apart from spring with the same ease.

- This has been achieved by dependency inversion and open closed principles.

- All modules expose only abstraction which is useful in extending the functionality or plug-in in another module.

Dependency Inversion Principle

- The Dependency Inversion Principle (DIP) states that high-level modules should not depend upon low-level modules; they should depend on abstractions.

- Secondly, abstractions should not depend upon details; details should depend upon abstractions.

- The idea is that we isolate our class behind a boundary formed by the abstractions it depends on.

- If all the details behind those abstractions change, then our class is still safe.

- This helps keep coupling low and makes our design easier to change.

- DIP also allows us to test things in isolation.

Code that violates Dependency Inversion Principle

- Suppose a book store asked us to build a new feature that enables customers to put their favorite books on a shelf.
- In order to implement the new functionality, we create a lower-level Book class and a higher-level Shelf class.
- The Book class will allow users to see reviews and read a sample of each book they store on their shelves.
- The Shelf class will let them add a book to their shelf and customize the shelf. For example, observe the code.

```
public class Book {

            void seeReviews()   {
                        ...
             }

            void readSample() {
                        ...
            }
}
  public class Shelf {
             Book book;
              void addBook(Book book) {
                    ...
              }
              void customizeShelf() {
                          ...
              }
}
```

- Everything looks fine, but as the high-level Shelf class depends on the low-level Book, the above code violates the Dependency Inversion Principle.

- This becomes clear when the store asks us to enable customers to add DVDs to their shelves, too.

- In order to fulfil the demand, we create a new DVD class:

```
public class DVD {

        void seeReviews() {
                ...
         }

         void watchSample() {
                ...
         }
}
```

- Now, we should modify the Shelf class so that it can accept DVDs, too.
- However, this would clearly break the Open/Closed Principle too.
- The solution is to create an abstraction layer for the lower-level classes (Book and DVD).
- We'll do so by introducing the Product interface, both classes will implement it.
- For example, this piece of code demonstrates the concept.

```
public interface Product {

void seeReviews();
void getSample();
}
public class Book implements Product {

@Override
public void seeReviews() {
...
 }
@Override
public void getSample() {
...
 }
}
 public class DVD implements Product {
@Override
 public void seeReviews() {
...
}
@Override
public void getSample() {
 ...
}
}
```

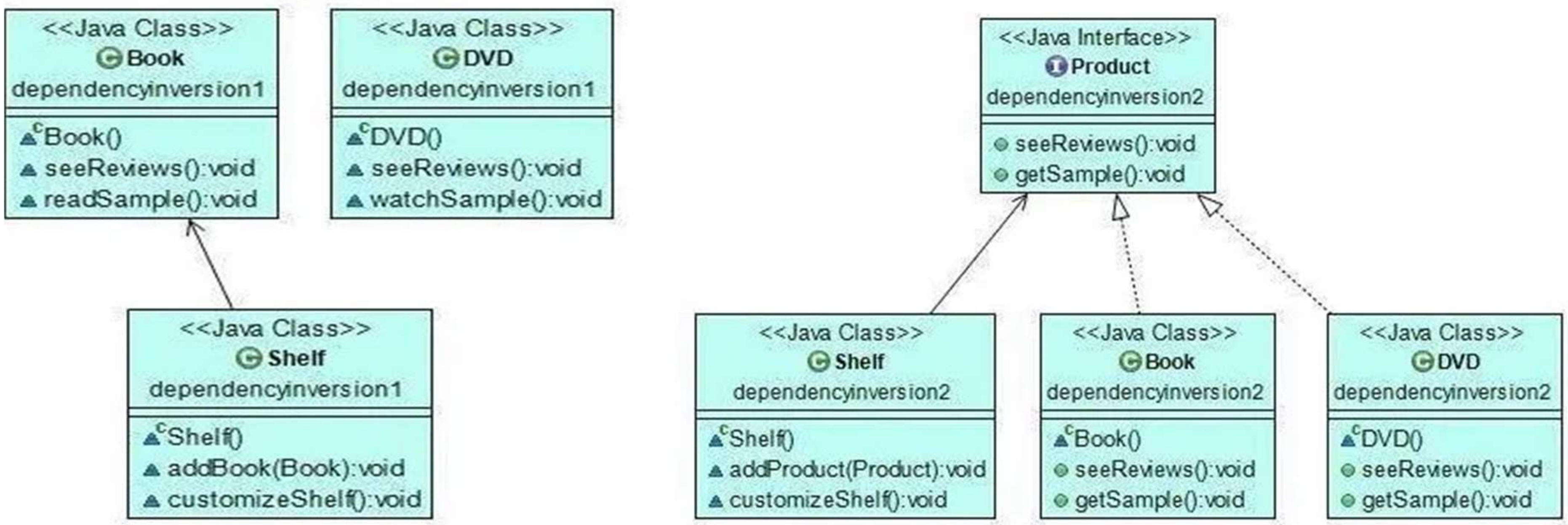
- Now, Shelf can reference the Product interface instead of its implementations (Book and DVD).

- The refactored code also allows us to later introduce new product types (for instance, Magazine) that customers can put on their shelves, too.

```
public class Shelf {

        Product product;

     void addProduct(Product product) {
                    ...
     }

     void customizeShelf() {
                    ...
     }
}
```

- The above code also follows the Liskov Substitution Principle, as the Product type can be substituted with both of its subtypes (Book and DVD) without breaking the program.
- At the same time, we have also implemented the Dependency Inversion Principle, as in the refactored code, high-level classes don’t depend on low-level classes, either.

Let’s check with the class diagram in both the cases.


<<Java Class>>
Book
dependencyinversion1
Book()
seeReviews():void
readSample():void
<<Java Class>>
DVD
dependencyinversion1
DVD()
seeReviews():void
watchSample():void
<<Java Class>>
Shelf
dependencyinversion1
Shelf()
addBook(Book):void
customizeShelf():void
<<Java Interface>>
Product
dependencyinversion2
seeReviews():void
getSample():void
<<Java Class>>
Shelf
dependencyinversion2
Shelf()
addProduct(Product):void
customizeShelf():void
<<Java Class>>
Book
dependencyinversion2
Book()
seeReviews():void
getSample():void
<<Java Class>>
DVD
dependencyinversion2
DVD()
seeReviews():void
getSample():void

Dependency Inversion Principle

Example

- A program depends on Reader and Writer interfaces that are abstractions, and Keyboard and Printer are details that depend on those abstractions by implementing those interfaces.

- Here CharCopier is oblivious to the low-level details of Reader and Writer implementations and thus you can pass in any Device that implements the Reader and Writer interface and CharCopier would still work correctly.

Dependency Inversion Principle Example

```
public interface Reader {
         char getchar();
}

public interface Writer {
          void putchar(char c);
}
 class CharCopier {
   void copy(Reader reader, Writer writer) {

      int c;

         while ((c = reader.getchar()) != EOF) {

             writer.putchar();

                     }
               }
          }
```

```
public Keyboard implements Reader
{
                   ...
}
public Printer implements Writer {
                   ...
}
```

Benefit of Dependency Inversion Principle

- some of the benefits when we apply this principle in our code.
- Keeps your code loosely coupled
- Easier Maintenance
- Better Code Reusability

BITS Pilani
Pilani Campus

Introduction to Design Patterns

A R Rahman

Introduction to Design Patterns

“Gang of Four” (GoF) Book:

- Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Publishing Company, 1994
- Written by this "gang of four“
  - Dr. Erich Gamma, then Software Engineer, Taligent, Inc.
  - Dr. Richard Helm, then Senior Technology Consultant, DMR Group
  - Dr. Ralph Johnson, then and now at University of Illinois, Computer Science Department
  - Dr. John Vlissides, then a researcher at IBM
    - Thomas J. Watson Research Center
    - See John's Wiki tribute page http://c2.com/cgi/wiki?JohnVlissides

The Beginning of Patterns

- Christopher Alexander, architect
  - A Pattern Language--Towns, Buildings, Construction
  - Timeless Way of Building (1979)
  - “Each pattern describes a *problem* which occurs over and over again in our environment, and then describes the core of the *solution* to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.”
- Other patterns: novels (tragic, romantic, crime), movies genres (drama, comedy, documentary)

“Gang of Four” (GoF) Book:

- This book defined 23 patterns in three categories
  - *Creational patterns* deal with the process of object creation
  - *Structural patterns*, deal primarily with the static composition and structure of classes and objects
  - *Behavioral patterns*, which deal primarily with dynamic interaction among classes and objects

“Gang of Four” Patterns :

- *Creational Patterns*
  - Abstract Factory
  - Builder
  - Factory Method
  - Prototype
  - Singleton

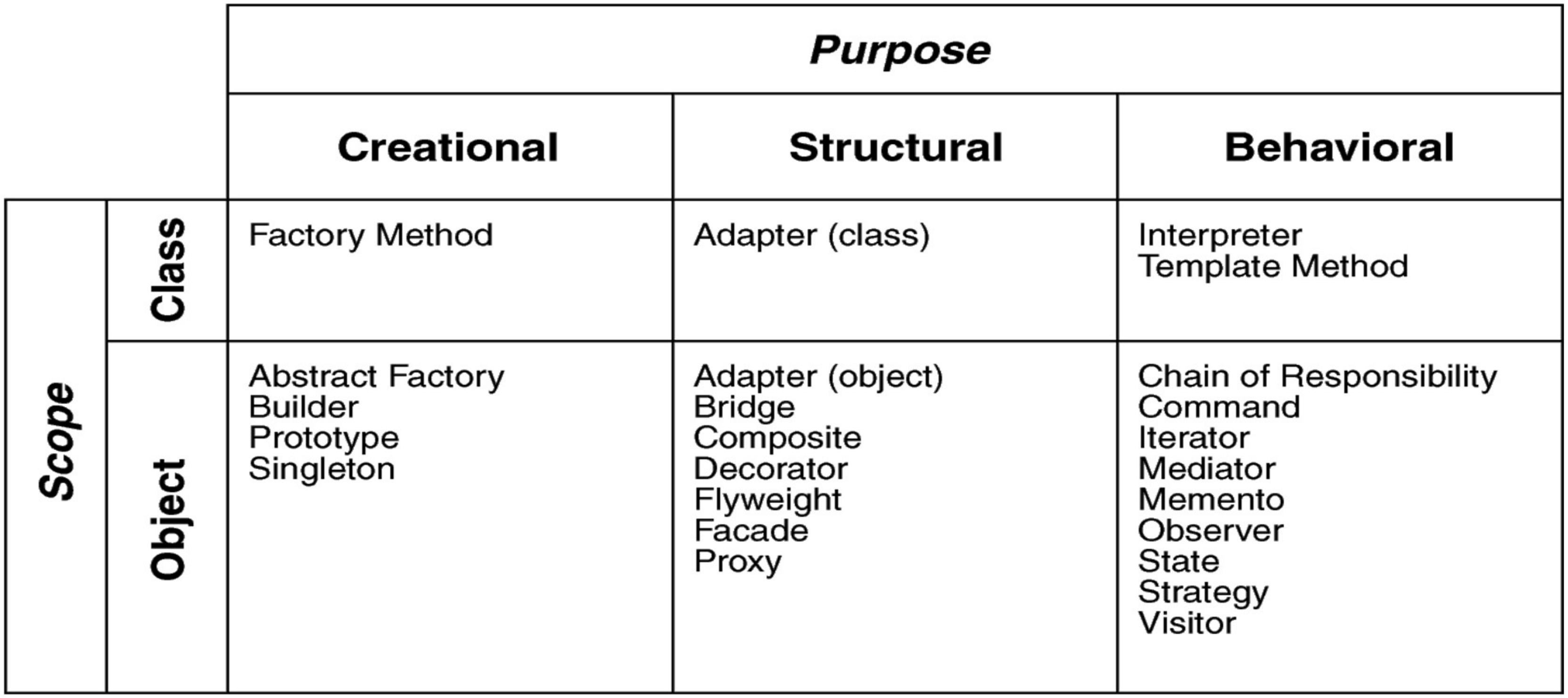
Structural Patterns

- Adapter
- Bridge
- Composite
- Decorator
- Façade
- Flyweight
- Proxy

“Gang of Four” Patterns :

- *Behavioral Patterns*
  - Chain of Responsibility
  - Command
  - Interpreter
  - Iterator
  - Mediator
  - Memento
  - Observer
  - State
  - Strategy
  - Template Method
  - Visitor

“Gang of Four” Patterns :



		Purpose		
		Creational	**Structural**	**Behavioral**
Scope	**Class**	Factory Method	Adapter (class)	Interpreter Template Method
	Object	Abstract Factory Builder Prototype Singleton	Adapter (object) Bridge Composite Decorator Flyweight Facade Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

Scope: domain over which a pattern applies

Purpose: reflects what a pattern does

GoF: Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides

Why Study Patterns?

- Reuse tried, proven solutions
  - Provides a head start

  - Avoids gotchas later (unanticipated things)

  - No need to reinvent the wheel

- Establish common terminology
  - Design patterns provide a common point of reference

  - Easier to say, “We could use Strategy here.”

- Provide a higher level prospective
  - Frees us from dealing with the details too early

Design patterns goals

- Codify good design
  - distill & generalize experience
  - aid to novices & experts alike
- Give design structures explicit names
  - common vocabulary
  - reduced complexity
  - greater expressiveness

Design patterns goals

- Capture & preserve design information
  - articulate design decisions succinctly
  - improve documentation
- Facilitate restructuring/refactoring
  - patterns are interrelated
  - additional flexibility

Design pattern template

- Intent
  - short description of the pattern & its purpose

- Also Known As
  - Any aliases this pattern is known by

- Motivation
  - motivating scenario demonstrating pattern’s use

- Applicability
  - circumstances in which pattern applies

Design pattern template

- Structure
    - graphical representation of the pattern using modified UML notation
- Participants
    - participating classes and/or objects & their responsibilities
- Collaborations
    - how participants cooperate to carry out their responsibilities
- Consequences
    - the results of application, benefits, liabilities

Design pattern template

- Implementation
    - pitfalls, hints, techniques, plus language-dependent issues

- Sample Code
    - sample implementations in C++, Java, C#, Smalltalk, C, etc.

- Known Uses
    - examples drawn from existing systems

- Related Patterns
    - discussion of other patterns that relate to this one

Other advantages

- Most design patterns make software more modifiable, less brittle
    - we are using time tested solutions
- Using design patterns makes software systems easier to change—more maintainable
- Helps increase the understanding of basic object-oriented design principles
    - encapsulation, inheritance, interfaces, polymorphism

Style for Describing Patterns

- We will use this structure:
  - *Pattern name*
  - *Recurring problem:* what problem the pattern addresses
  - *Solution:* the general approach of the pattern
  - *UML for the pattern*
    - *Participants:* a description as a class diagram
  - *Use Example(s):* examples of this pattern, in Java

A few OO Design Patterns

- Coming up:
  - Iterator
    - access the elements of an aggregate object sequentially without exposing its underlying representation
  - Strategy
    - A means to define a family of algorithms, encapsulate each one as an object, and make them interchangeable
  - Observer *a preview*
    - One object stores a list of observers that are updated when the state of the object is changed
  - Adaptor
    - *Adapter* changes interface of one so the other can use it

The Singleton Design Pattern

The Singleton Design Pattern

- The singleton pattern only allows a class or object to have a single instance and it uses a global variable to store that instance.

- You can use lazy loading to make sure that there is only one instance of the class because it will only create the class when you need it.

- That prevents multiple instances from being active at the same time which could cause weird bugs.

- Most of the time this gets implemented in the constructor. The goal of the singleton pattern is typically to regulate the global state of an application.

The Singleton Design Pattern

- The singleton pattern is one of the simplest design patterns.
- Sometimes we need to have only one instance of our class for example a single DB connection shared by multiple objects as creating a separate DB connection for every object may be costly.
- Similarly, there can be a single configuration manager or error manager in an application that handles all problems instead of creating multiple managers.
- Definition:
  The singleton pattern is a design pattern that restricts the instantiation of a class to one object.
- Let’s see various design options for implementing such a class.
- If you have a good handle on static class variables and access modifiers this should not be a difficult task.

Method 1: Classic Implementation

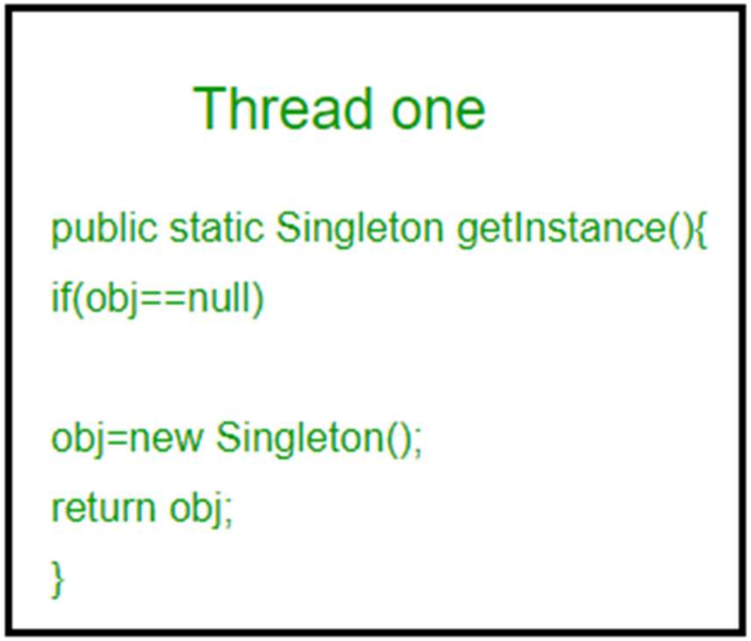
// Classical Java implementation of singleton design pattern

```
•   class Singleton
•   {
•       private static Singleton obj;
•
•       // private constructor to force use of
•       // getInstance() to create Singleton object
•       private Singleton() {}

•       public static Singleton getInstance()
•       {
•           if (obj==null)
•               obj = new Singleton();
•           return obj;
•       }
•   }
```

Method 1: Classic Implementation

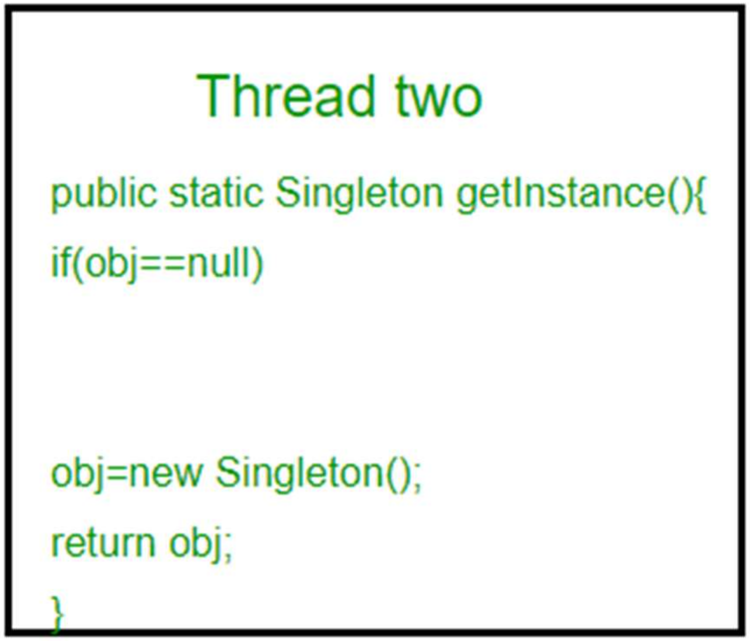
- Here we have declared getInstance() static so that we can call it without instantiating the class.
- The first time getInstance() is called it creates a new singleton object and after that it just returns the same object.
- Note that Singleton obj is not created until we need it and call getInstance() method.
- This is called lazy instantiation.
- The main problem with above method is that it is not thread safe.
- Consider the following execution sequence.

Method 1: Classic Implementation



Thread one

```
public static Singleton getInstance(){
if(obj==null)

obj=new Singleton();
return obj;
}
```



Thread two

```
public static Singleton getInstance(){
if(obj==null)


obj=new Singleton();
return obj;
}
```

- This execution sequence creates two objects for singleton.
- Therefore this classic implementation is not thread safe.

Method 2: make getInstance() synchronized

```
// Thread Synchronized Java implementation of singleton design pattern

class Singleton
{
   private static Singleton obj;

   private Singleton() {}

   // Only one thread can execute this at a time
   public static synchronized Singleton getInstance()
   {
      if (obj==null)
         obj = new Singleton();
      return obj;
   }
}
```

Method 2: make getInstance() synchronized

- Here using synchronized makes sure that only one thread at a time can execute getInstance().

- The main disadvantage of this is method is that using synchronized every time while creating the singleton object is expensive and may decrease the performance of your program.

- However if performance of getInstance() is not critical for your application this method provides a clean and simple solution.

Method 3: Eager Instantiation

```
// Static initializer based Java implementation of  singleton design pattern

class Singleton
{
    private static Singleton obj = new Singleton();

    private Singleton() {}

    public static Singleton getInstance()
    {
        return obj;
    }
}
```

Method 3: Eager Instantiation

- Here we have created instance of singleton in static initializer.
- JVM executes static initializer when the class is loaded and hence this is guaranteed to be thread safe.
- Use this method only when your singleton class is light and is used throughout the execution of your program.

Method 4 (Best): Use “Double Checked Locking”

- If you notice carefully once an object is created synchronization is no longer useful because now obj will not be null and any sequence of operations will lead to consistent results.

- So we will only acquire lock on the getInstance() once, when the obj is null.

- This way we only synchronize the first way through, just what we want.

- We have declared the obj volatile which ensures that multiple threads offer the obj variable correctly when it is being initialized to Singleton instance.

- This method drastically reduces the overhead of calling the synchronized method every time.

```
// Double Checked Locking based Java implementation of singleton design pattern
 class Singleton
 {
     private static volatile Singleton obj  = null;

     private Singleton() {}

     public static Singleton getInstance()
     {
         if (obj == null)
         {
             // To make thread safe
             synchronized (Singleton.class)
             {
                 // check again as multiple threads
                 // can reach above step
                 if (obj==null)
                     obj = new Singleton();
             }
         }
         return obj;
     }
 }
```

The Singleton Design Pattern

- An example of a singleton that you probably use all the time is your logger.
- If you work with some of the front-end frameworks like React or Angular, you know all about how tricky it can be to handle logs coming from multiple components.
- This is a great example of singletons in action because you never want more than one instance of a logger object, especially if you're using some kind of error tracking tool.

```
class FoodLogger {
  constructor() {
    this.foodLog = []
  }

  log(order) {
    this.foodLog.push(order.foodItem)
    // do fancy code to send this log somewhere
  }
}
// this is the singleton
class FoodLoggerSingleton {
  constructor() {
    if (!FoodLoggerSingleton.instance) {
      FoodLoggerSingleton.instance = new FoodLogger()
    }
  }
getFoodLoggerInstance() {
    return FoodLoggerSingleton.instance
  }
}

module.exports = FoodLoggerSingleton
```

An example of the singleton class

The Singleton Design Pattern

- Now you don't have to worry about losing logs from multiple instances because you only have one in your project.

- So when you want to log the food that has been ordered, you can use the same *FoodLogger* instance across multiple files or components.

```
• const FoodLogger = require('./FoodLogger')

• const foodLogger = new FoodLogger().getFoodLoggerInstance()

• class Customer {
•   constructor(order) {
•     this.price = order.price
•     this.food = order.foodItem
•     foodLogger.log(order)
•   }
•
•   // other cool stuff happening for the customer
• }

• module.exports = Customer
```

An example of a Customer class using the singleton

```
• const FoodLogger = require('./FoodLogger')

• const foodLogger = new FoodLogger().getFoodLoggerInstance()

• class Restaurant {
•   constructor(inventory) {
•     this.quantity = inventory.count
•     this.food = inventory.foodItem
•     foodLogger.log(inventory)
•   }
•
•   // other cool stuff happening at the restaurant
• }

• module.exports = Restaurant
```

An example of the Restaurant class using the same singleton as the Customer class

- With this singleton pattern in place, you don't have to worry about just getting the logs from the main application file.

- You can get them from anywhere in your code base and they will all go to the exact same instance of the logger, which means none of your logs should get lost due to new instances.

The Strategy Design Pattern

Pattern: *Strategy*

- **Name**: Strategy (a.k.a Policy)
- **Problem**: You want to encapsulate a family of algorithms and make them interchangeable.
- Strategy lets the algorithm vary independently from the clients that use it (GoF)
- **Solution**: Create an abstract strategy class (or interface) and extend (or implement) it in numerous ways.
- Each subclass defines the same method names in different ways

Pattern: *Strategy*

- Consequences:
  - Allows families of algorithms
- Known uses:
  - Critters seen in section for Rick’s 127B / 227
  - Layout managers in Java
  - Different Poker Strategies in a 335 Project
  - Different PacMan chase strategies in a 335 Project
  - Different Jukebox policies that can be planned

Pattern: *Strategy*

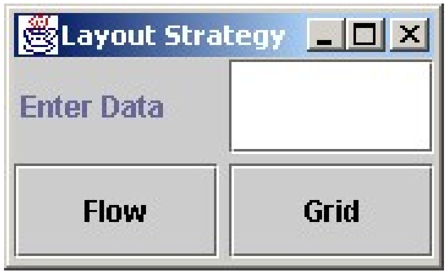
```
this.setLayout(new FlowLayout());

this.setLayout(new GridLayout());
```

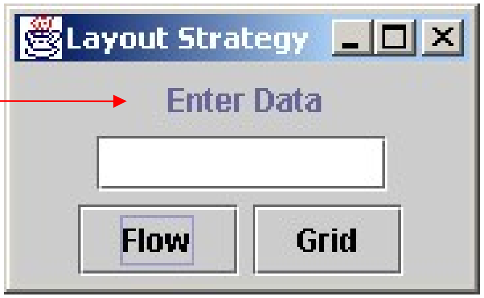
- In Java, a container HAS-A layout manager
  - There is a default
  - You can change a container's layout manager with a setLayout message

Change the strategy at runtime

- LayoutControllerFrame.java

```
   private class FlowListener
        implements ActionListener {

 // There is another ActionListener for GridLayout
 public void actionPerformed(ActionEvent evt) {

  // Change the layout strategy of the JPanel
  // and tell it to lay itself out

  centerPanel.setLayout(new FlowLayout());

  centerPanel.validate();

 }
}
```


Layout Strategy
Enter Data
Flow
Grid


Layout Strategy
Enter Data
Flow
Grid

interface LayoutManager

- Java has interface java.awt.LayoutManager

- Known Implementing Classes
  - GridLayout, FlowLayout, ScrollPaneLayout

- Each class implements the following methods

addLayoutComponent(String name, Component comp)

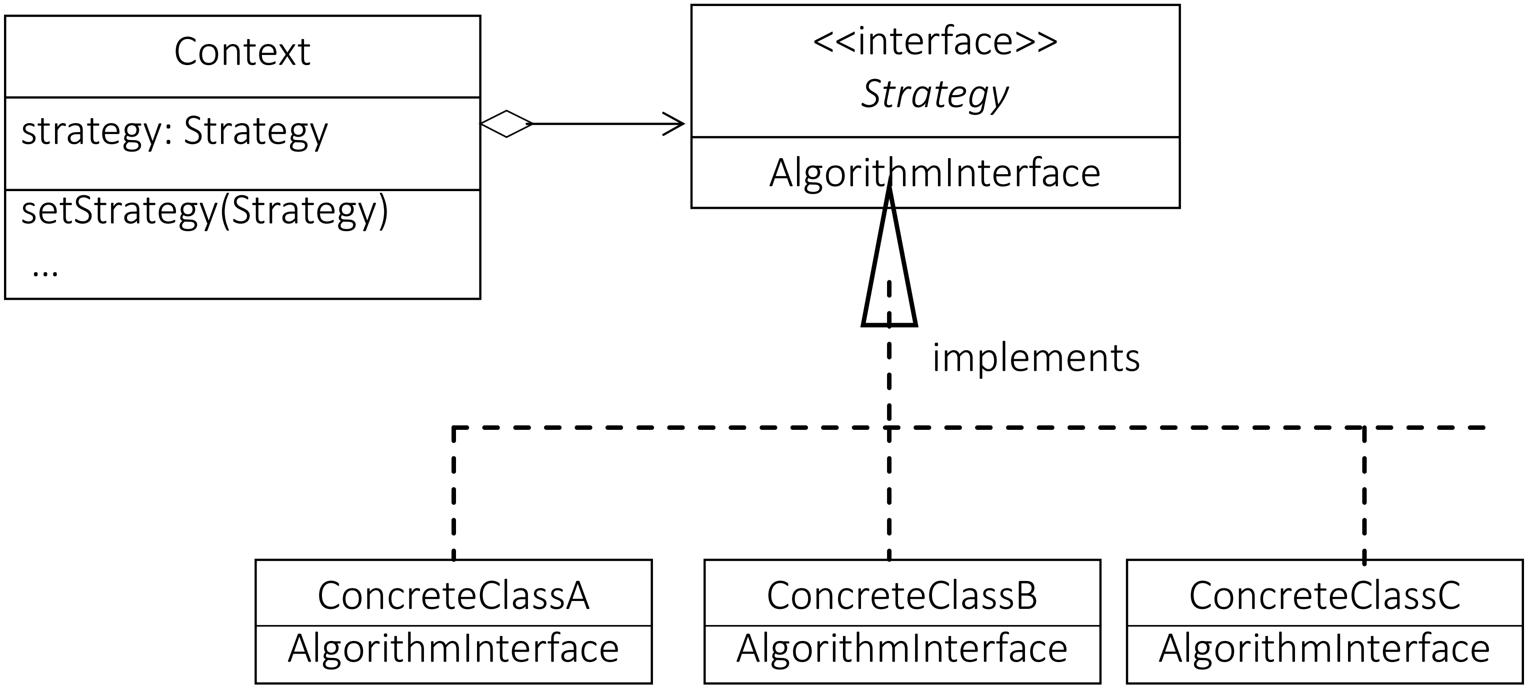
layoutContainer(Container parent)

minimumLayoutSize(Container parent)

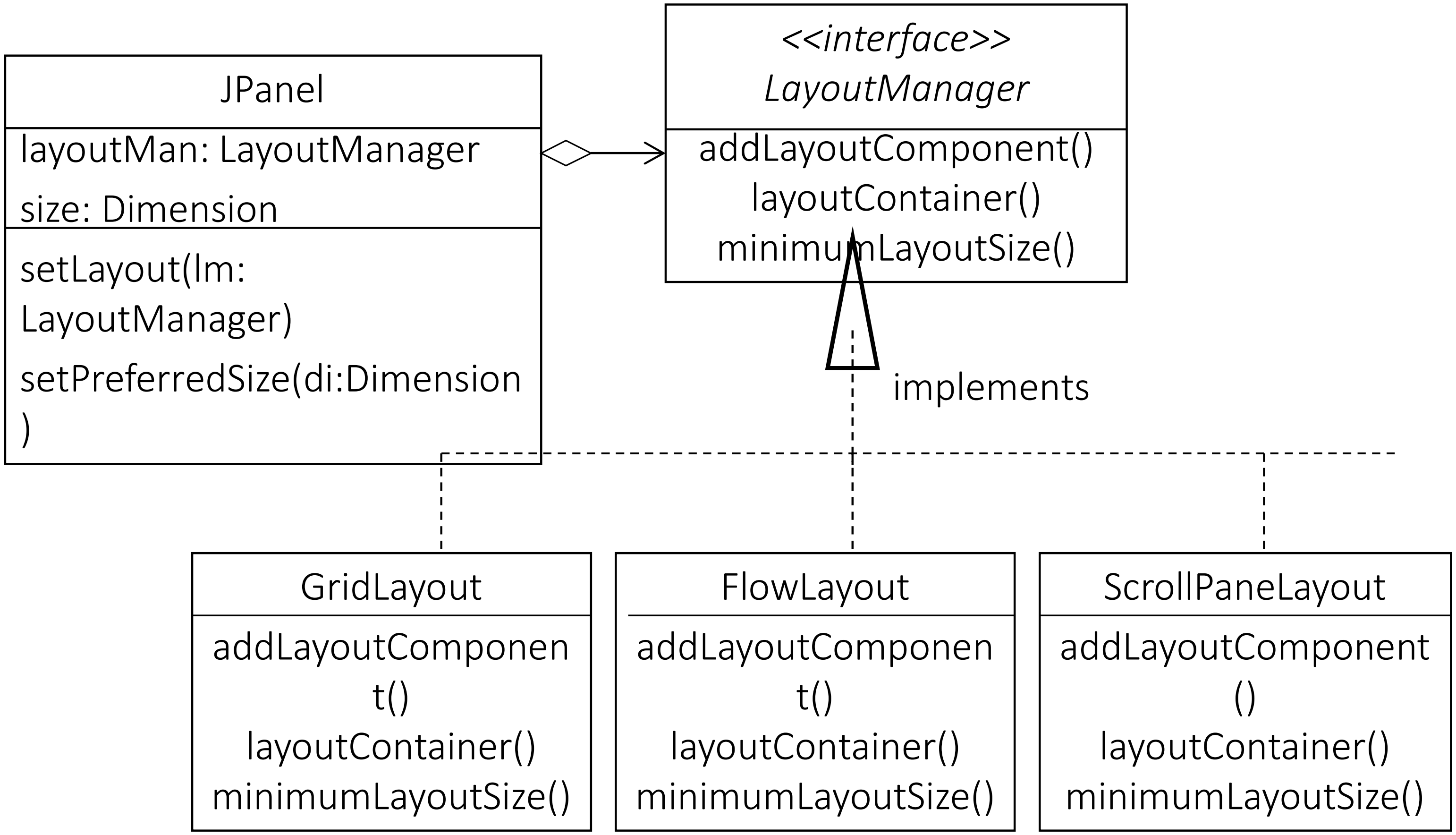
preferredLayoutSize(Container parent)

removeLayoutComponent(Component comp)

UML Diagram of Strategy General Form


Context
strategy: Strategy
setStrategy(Strategy)
…
<<interface>>
Strategy
AlgorithmInterface
implements
ConcreteClassA
AlgorithmInterface
ConcreteClassB
AlgorithmInterface
ConcreteClassC
AlgorithmInterface

Specific UML Diagram of LayoutManager in Java



JPanel
layoutMan: LayoutManager
size: Dimension
setLayout(lm: LayoutManager)
setPreferredSize(di:Dimension)
<<interface>>
LayoutManager
addLayoutComponent()
layoutContainer()
minimumLayoutSize()
implements
GridLayout
addLayoutComponent()
layoutContainer()
minimumLayoutSize()
FlowLayout
addLayoutComponent()
layoutContainer()
minimumLayoutSize()
ScrollPaneLayout
addLayoutComponent()
layoutContainer()
minimumLayoutSize()

Another Example

- Pac Man GhostChasesPacMan strategies in 2001
  - Level 1: random
  - Level 2: a bit smarter
  - Level 3: use a shortest path algorithm
  - http://www.martystepp.com/applets/pacman/
- Could be interface ChaseStrategy is in the Ghost class

```
interface ChaseStategy {
 public Point nextPointToMoveTo();
}
```

- and Ghost has setChaseStrategy(new ShortestPath())

The Strategy Design Pattern

- The strategy is pattern is like an advanced version of an if else statement.
- It's basically where you make an interface for a method you have in your base class.
- This interface is then used to find the right implementation of that method that should be used in a derived class.
- The implementation, in this case, will be decided at runtime based on the client.
- This pattern is incredibly useful in situations where you have required and optional methods for a class.

The Strategy Design Pattern

- Some instances of that class won't need the optional methods, and that causes a problem for inheritance solutions.
- You could use interfaces for the optional methods, but then you would have to write the implementation every time you used that class since there would be no default implementation.
- That's where the strategy pattern saves us. Instead of the client looking for an implementation, it delegates to a strategy interface and the strategy finds the right implementation.
- One common use for this is with payment processing systems.

The Strategy Design Pattern

- You *could* have a shopping cart that only lets customers check out with their credit cards, but you will lose customers that want to use other payment methods.

- The strategy design pattern lets us decouple the payment methods from the checkout process which means we can add or update strategies without changing any code in the shopping cart or checkout process.

- Here's an example of a strategy pattern implementation using the payment method example.

- class PaymentMethodStrategy {
- const customerInfoType = {
- country: string
- emailAddress: string
- name: string
- accountNumber?: number
- address?: string
- cardNumber?: number
- city?: string
- routingNumber?: number
- state?: string
- }
- static BankAccount(customerInfo: customerInfoType) {
- const { name, accountNumber, routingNumber } = customerInfo
- // do stuff to get payment
- }

- static BitCoin(customerInfo: customerInfoType) {
-   const { emailAddress, accountNumber } = customerInfo
-   // do stuff to get payment
- }
-
- static CreditCard(customerInfo: customerInfoType) {
-   const { name, cardNumber, emailAddress } = customerInfo
-   // do stuff to get payment
- }

```
• static MailIn(customerInfo: customerInfoType) {
•     const { name, address, city, state, country } = customerInfo
•     // do stuff to get payment
•   }
• 
•   static PayPal(customerInfo: customerInfoType) {
•     const { emailAddress } = customerInfo
•     // do stuff to get payment
•   }
• }
```

- To implement our payment method strategy, we made a single class with multiple static methods.
- Each method takes the same parameter, *customerInfo*, and that parameter has a defined type of *customerInfoType*.
- (Hey all you TypeScript devs! ??) Take note that each method has its own implementation and uses different values from the *customerInfo*.
- With the strategy pattern, you can also dynamically change the strategy being used at run time.
- That means you'll be able to change the strategy, or method implementation, being used based on user input or the environment the app is running in.
- You can also set a default implementation in a simple *config.json* file like this:

- config.json for setting the default implementation of paymentMethod to "PayPal" at run time

- {
-   "paymentMethod": {
-     "strategy": "PayPal"
-   }
- }

- Whenever a customer starts going through the checkout process on your website, the default payment method they encounter will be the PayPal implementation which comes from the *config.json*.

- This could easily be updated if the customer selects a different payment method.

- Now we'll create a file for our checkout process.

```
const PaymentMethodStrategy = require('./PaymentMethodStrategy')
const config = require('./config')

class Checkout {
  constructor(strategy='CreditCard') {
    this.strategy = PaymentMethodStrategy[strategy]
  }

  // do some fancy code here and get user input and payment method

  changeStrategy(newStrategy) {
    this.strategy = PaymentMethodStrategy[newStrategy]
  }
• const userInput = {
•     name: ‘Rahman',
•     cardNumber: 3910000034581941,
•     emailAddress: ‘ar@gmailer.com',
•     country: 'US'
•   }
•     const selectedStrategy = 'Bitcoin'
•
•   changeStrategy(selectedStrategy)
•
•   postPayment(userInput) {
•     this.strategy(userInput)
•   }
• }
• module.exports = new Checkout(config.paymentMethod.strategy)
```

- This *Checkout* class is where the strategy pattern gets to show off.
- We import a couple of files so we have the payment method strategies available and the default strategy from the *config*.
- Then we create the class with the constructor and a fallback value for the default *strategy* in case there hasn't been one set in the *config*.
- Next we assign the *strategy* value to a local state variable.
- An important method we need to implement in our *Checkout* class is the ability to change the payment strategy.
- A customer might change the payment method they want to use and you'll need to be able to handle that. That's what the *changeStrategy* method is for.

- After you've done some fancy coding and gotten all of the inputs from a customer, then you can update the payment strategy immediately based on their input and it dynamically sets the *strategy* before the payment is sent for processing.

- At some point you might need to add more payment methods to your shopping cart and all you'll have to do is add it to the *PaymentMethodStrategy* class. It'll instantly be available anywhere that class is used.

- The strategy design pattern is a powerful one when you are dealing with methods that have multiple implementations.

- It might feel like you're using an interface, but you don't have to write an implementation for the method every time you call it in a different class. It gives you more flexibility than interfaces.

The Observer Design Pattern

Observer design pattern

Intent

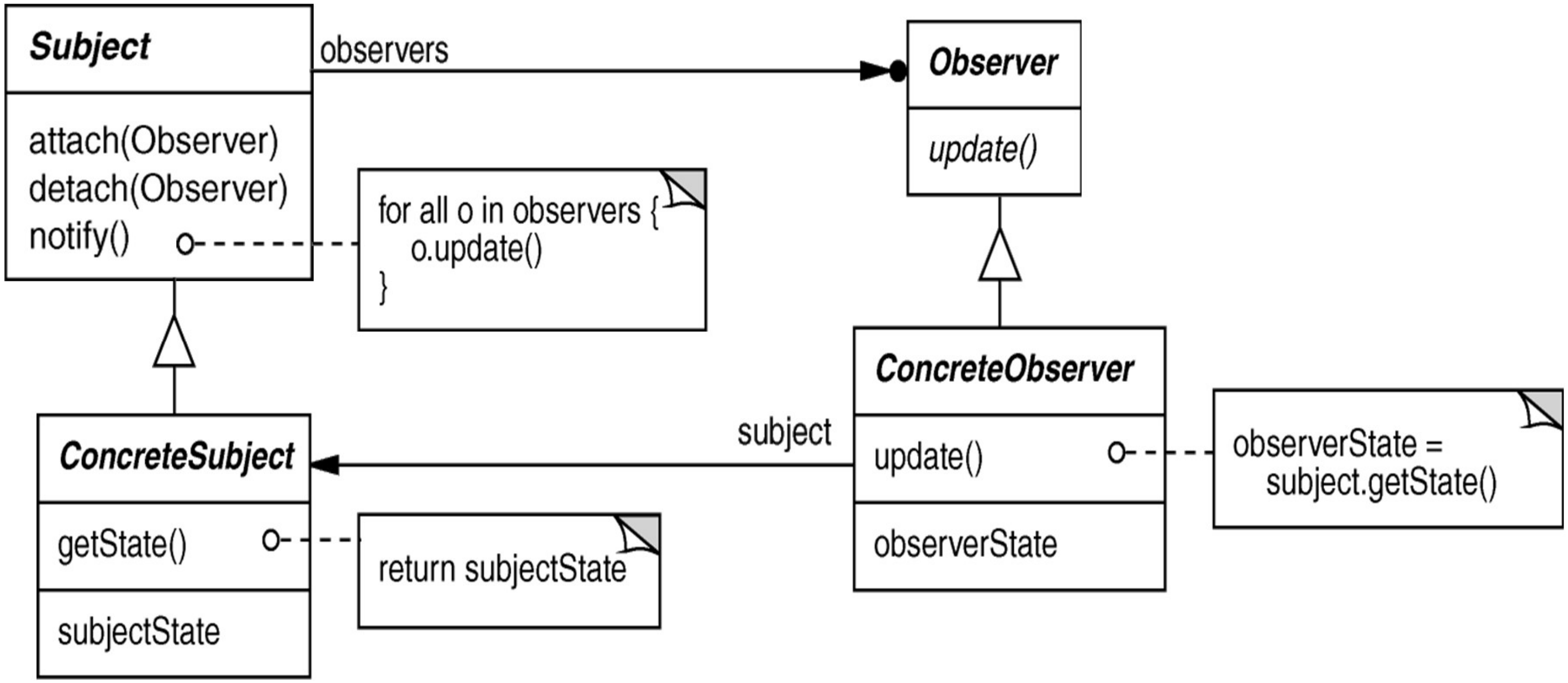
- define a one-to-many dependency between objects so that when one object changes state, all dependents are notified & updated

Applicability

- an abstraction has two aspects, one dependent on the other
- a change to one object requires changing untold others
- an object should notify unknown other objects

Observer design pattern

Structure


Subject
attach(Observer)
detach(Observer)
notify()
observers
Observer
update()
for all o in observers {
o.update()
}
ConcreteSubject
getState()
subjectState
return subjectState
subject
ConcreteObserver
update()
observerState
observerState =
subject.getState()

Observer design pattern

- class ProxyPushConsumer : public // …
- virtual void push (const CORBA::Any &event) {
- for (std::vector<PushConsumer>::iterator i
- (consumers.begin ()); i != consumers.end (); i++)
- (*i).push (event);
- }
- class MyPushConsumer : public // ….
- virtual void push
- (const CORBA::Any &event) { /* consume the event. */ }

Observer design pattern

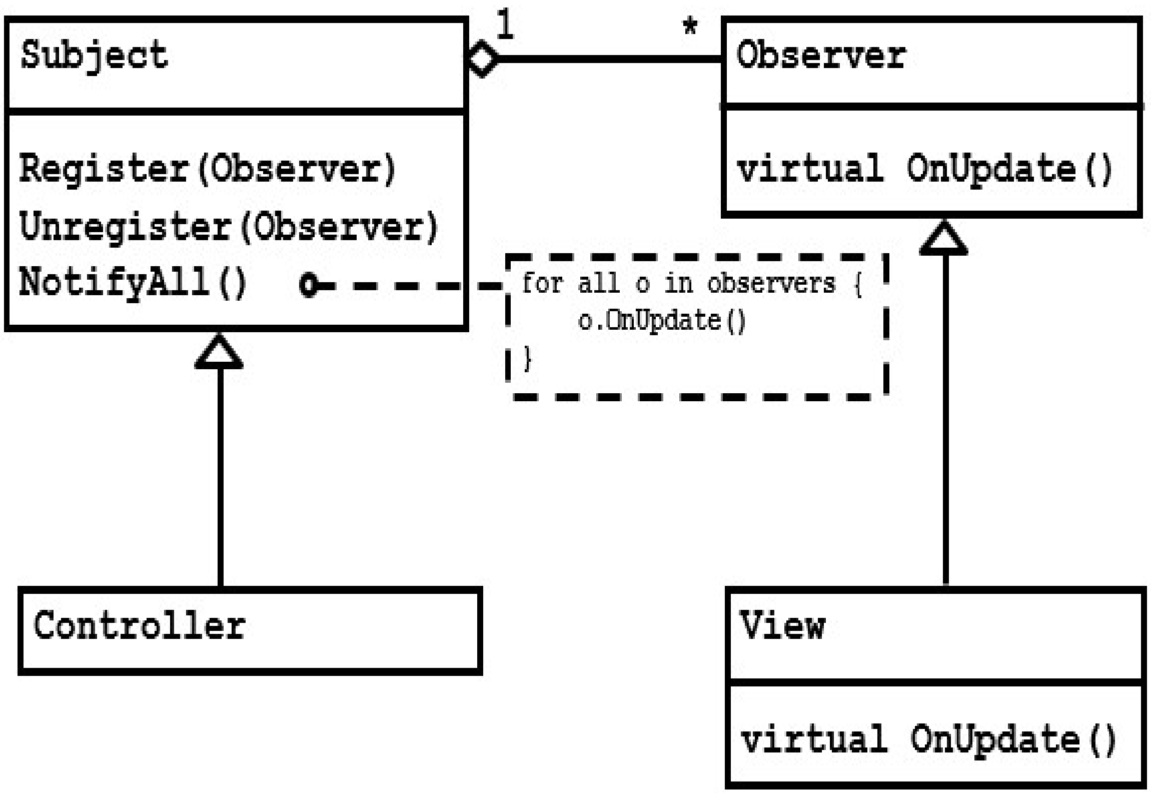
- Consequences
  - *modularity: subject & observers may vary independently*
  - *extensibility: can define & add any number of observers*
  - *customizability: different observers offer different views of subject*
  - unexpected updates: observers don’t know about each other
  - update overhead: might need hints or filtering

Observer design pattern

- Implementation
    - subject-observer mapping
    - dangling references
    - update protocols: the push & pull models
    - registering modifications of interest explicitly

Observer design pattern

- Known Uses
  - Smalltalk Model-View-Controller (MVC)
  - InterViews (Subjects & Views, Observer/Observable)
  - Pub/sub middleware
  - (e.g., CORBA Notification Service,
  - Java Messaging Service)
  - Mailing lists


1
*
Subject
Register(Observer)
Unregister(Observer)
NotifyAll()
Observer
virtual OnUpdate()
for all o in observers {
o.OnUpdate()
}
Controller
View
virtual OnUpdate()

The Observer Design Pattern

- If you've ever used the MVC pattern, you've already used the observer design pattern.

- The Model part is like a subject and the View part is like an observer of that subject. Your subject holds all of the data and the state of that data.

- Then you have observers, like different components, that will get that data from the subject when the data has been updated.

- The goal of the observer design pattern is to create this one-to-many relationship between the subject and all of the observers waiting for data so they can be updated.

The Observer Design Pattern

- So anytime the state of the subject changes, all of the observers will be notified and updated instantly.
- Some examples of when you would use this pattern include: sending user notifications, updating, filters, and handling subscribers.
- Say you have a single page application that has three feature dropdown lists that are dependent on the selection of a category from a higher level dropdown.
- This is common on many shopping sites, like Home Depot. You have a bunch of filters on the page that are dependent on the value of a top-level filter.
- The code for the top-level dropdown might look something like this:

```
• class CategoryDropdown {
•   constructor() {
•     this.categories = ['appliances', 'doors', 'tools']
•     this.subscriber = []
•   }
•
•   // pretend there's some fancy code here
•
•   subscribe(observer) {
•     this.subscriber.push(observer)
•   }
•
•   onChange(selectedCategory) {
•     this.subscriber.forEach(observer => observer.update(selectedCategory))
•   }
• }
```

The subject that updates the observers

- This *CategoryDropdown* file is a simple class with a constructor that initializes the category options we have available for in the dropdown.
- This is the file you would handle retrieving a list from the back-end or any kind of sorting you want to do before the user sees the options.
- The *subscribe* method is how each filter created with this class will receive updates about the state of the observer.
- The *onChange* method is how we send out notification to all of the subscribers that a state change has happened in the observer they're watching.
- We just loop through all of the subscribers and call their *update* method with the *selectedCategory*.
- The code for the other filters might look something like this:

```
• class FilterDropdown {
•   constructor(filterType) {
•     this.filterType = filterType
•     this.items = []
•   }
• 
•   // more fancy code here; maybe make that API call to get items list based on filterType
• 
•   update(category) {
•     fetch('https://example.com')
•       .then(res => this.items(res))
•   }
• }
```

A potential observer of the subject

- This *FilterDropdown* file is another simple class that represents all of the potential dropdowns we might use on a page.

- When a new instance of this class is created, it needs to be passed a *filterType.*

- This could be used to make specific API calls to get the list of items.

- The *update* method is an implementation of what you can do with the new category once it has been sent from the observer.

- Now we'll take a look at what it means to use these files with the observer pattern:

- const CategoryDropdown = require('./CategoryDropdown')
- const FilterDropdown = require('./FilterDropdown')

- const categoryDropdown = new CategoryDropdown()

- const colorsDropdown = new FilterDropdown('colors')
- const priceDropdown = new FilterDropdown('price')
- const brandDropdown = new FilterDropdown('brand')

- categoryDropdown.subscribe(colorsDropdown)
- categoryDropdown.subscribe(priceDropdown)
- categoryDropdown.subscribe(brandDropdown)

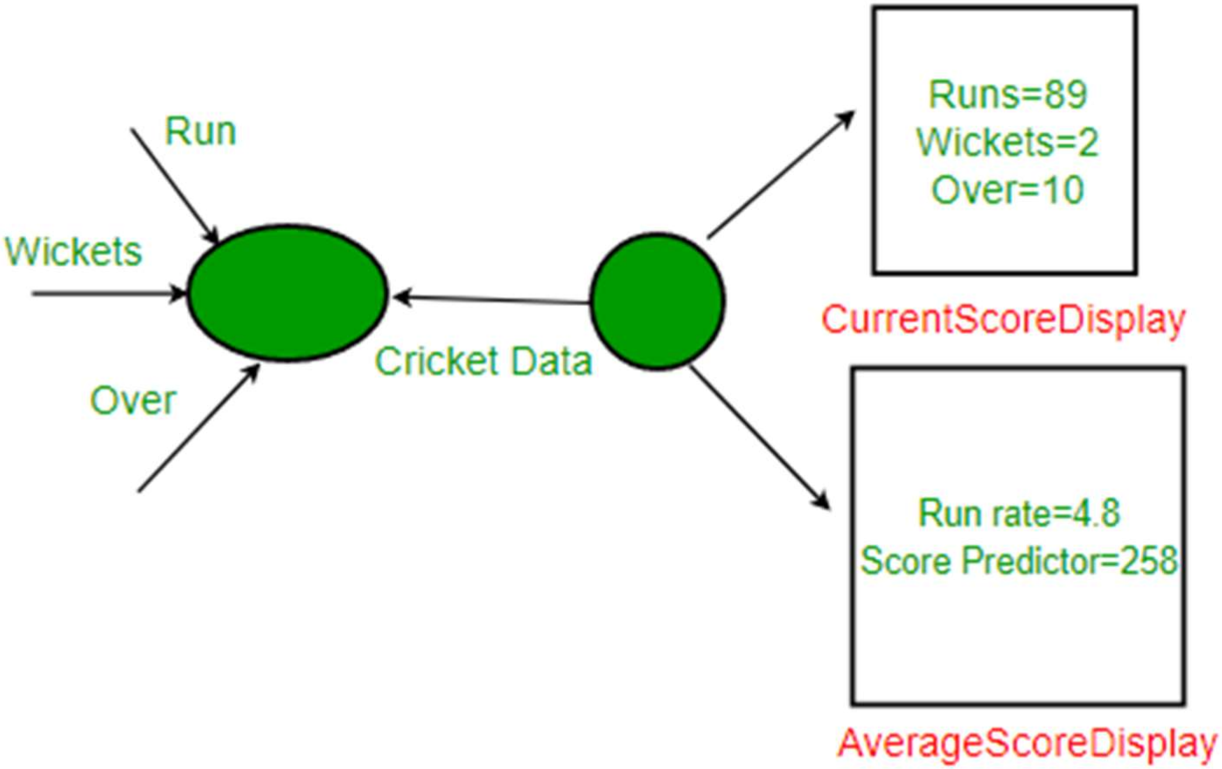
The subject that updates the observers

- What this file shows us is that we have 3 drop-downs that are subscribers to the category drop-down observable.

- Then we subscribe each of those drop-downs to the observer.

- Whenever the category of the observer is updated, it will send out the value to every subscriber which will update the individual drop-down lists instantly.

- Let us first consider the following scenario to understand observer pattern.
- Scenario:
- Suppose we are building a cricket app that notifies viewers about the information such as current score, run rate etc.
- Suppose we have made two display elements CurrentScoreDisplay and AverageScoreDisplay. CricketData has all the data (runs, balls etc.) and whenever data changes the display elements are notified with new data and they display the latest data accordingly.


Run
Wickets
Over
Cricket Data
Runs=89
Wickets=2
Over=10
CurrentScoreDisplay
Run rate=4.8
Score Predictor=258
AverageScoreDisplay

```
// Java implementation of above design for Cricket App. The problems with this design are discussed below.
// A class that gets information from stadium and notifies display elements, CurrentScoreDisplay & AverageScoreDisplay
class CricketData
{
    int runs, wickets;
    float overs;
    CurrentScoreDisplay currentScoreDisplay;
    AverageScoreDisplay averageScoreDisplay;

    // Constructor
    public CricketData(CurrentScoreDisplay currentScoreDisplay,
                       AverageScoreDisplay averageScoreDisplay)
    {
        this.currentScoreDisplay = currentScoreDisplay;
        this.averageScoreDisplay = averageScoreDisplay;
    }

    // Get latest runs from stadium
    private int getLatestRuns()
    {
        // return 90 for simplicity
        return 90;
    }
```

```
// Get latest wickets from stadium

   private int getLatestWickets()
   {
      // return 2 for simplicity
      return 2;
   }

   // Get latest overs from stadium

   private float getLatestOvers()
   {
      // return 10.2 for simplicity
      return (float)10.2;
   }
```

```
// This method is used update displays when data changes

   public void dataChanged()
   {
      // get latest data
      runs = getLatestRuns();
      wickets = getLatestWickets();
      overs = getLatestOvers();

      currentScoreDisplay.update(runs,wickets,overs);
      averageScoreDisplay.update(runs,wickets,overs);
   }
}
```

```
// A class to display average score. Data of this class is updated by CricketData

class AverageScoreDisplay
{
   private float runRate;
   private int predictedScore;

   public void update(int runs, int wickets, float overs)
   {
      this.runRate = (float)runs/overs;
      this.predictedScore = (int) (this.runRate * 50);
      display();
   }

   public void display()
   {
      System.out.println("\nAverage Score Display:\n" +
                         "Run Rate: " + runRate +
                         "\nPredictedScore: " + predictedScore);
   }
}
```

```
// A class to display score. Data of this class is // updated by CricketData
class CurrentScoreDisplay
{
    private int runs, wickets;
    private float overs;

    public void update(int runs,int wickets,float overs)
    {
        this.runs = runs;
        this.wickets = wickets;
        this.overs = overs;
        display();
    }

    public void display()
    {
        System.out.println("\nCurrent Score Display: \n" +
                           "Runs: " + runs +"\nWickets:"
                           + wickets + "\nOvers: " + overs );
    }
}
```

```
// Driver class
class Main
{
    public static void main(String args[])
    {
        // Create objects for testing
        AverageScoreDisplay averageScoreDisplay =
                                new AverageScoreDisplay();
        CurrentScoreDisplay currentScoreDisplay =
                                new CurrentScoreDisplay();

        // Pass the displays to Cricket data
        CricketData cricketData = new CricketData(currentScoreDisplay,
                                       averageScoreDisplay);

        // In real app you would have some logic to call this function when data changes
        cricketData.dataChanged();
    }
}
```

Output:

```
Current Score Display:
Runs: 90
Wickets:2
Overs: 10.2

Average Score Display:
Run Rate: 8.823529
PredictedScore: 441
```

- Problems with above design:

- CricketData holds references to concrete display element objects even though it needs to call only the update method of these objects.

- It has access to too much additional information than it requires.

- This statement “currentScoreDisplay.update(runs,wickets,overs);” violates one of the most important design principle “Program to interfaces, not implementations.” as we are using concrete objects to share data rather than abstract interfaces.

- CricketData and display elements are tightly coupled.

- If in future another requirement comes in and we need another display element to be added we need to make changes to the non-varying part of our code(CricketData).

- This is definitely not a good design practice and application might not be able to handle changes and not easy to maintain.

- How to avoid these problems?
- Use Observer Pattern

- Observer pattern

- To understand observer pattern, first you need to understand the subject and observer objects.

- The relation between subject and observer can easily be understood as an analogy to magazine subscription.

- A magazine publisher(subject) is in the business and publishes magazines (data).

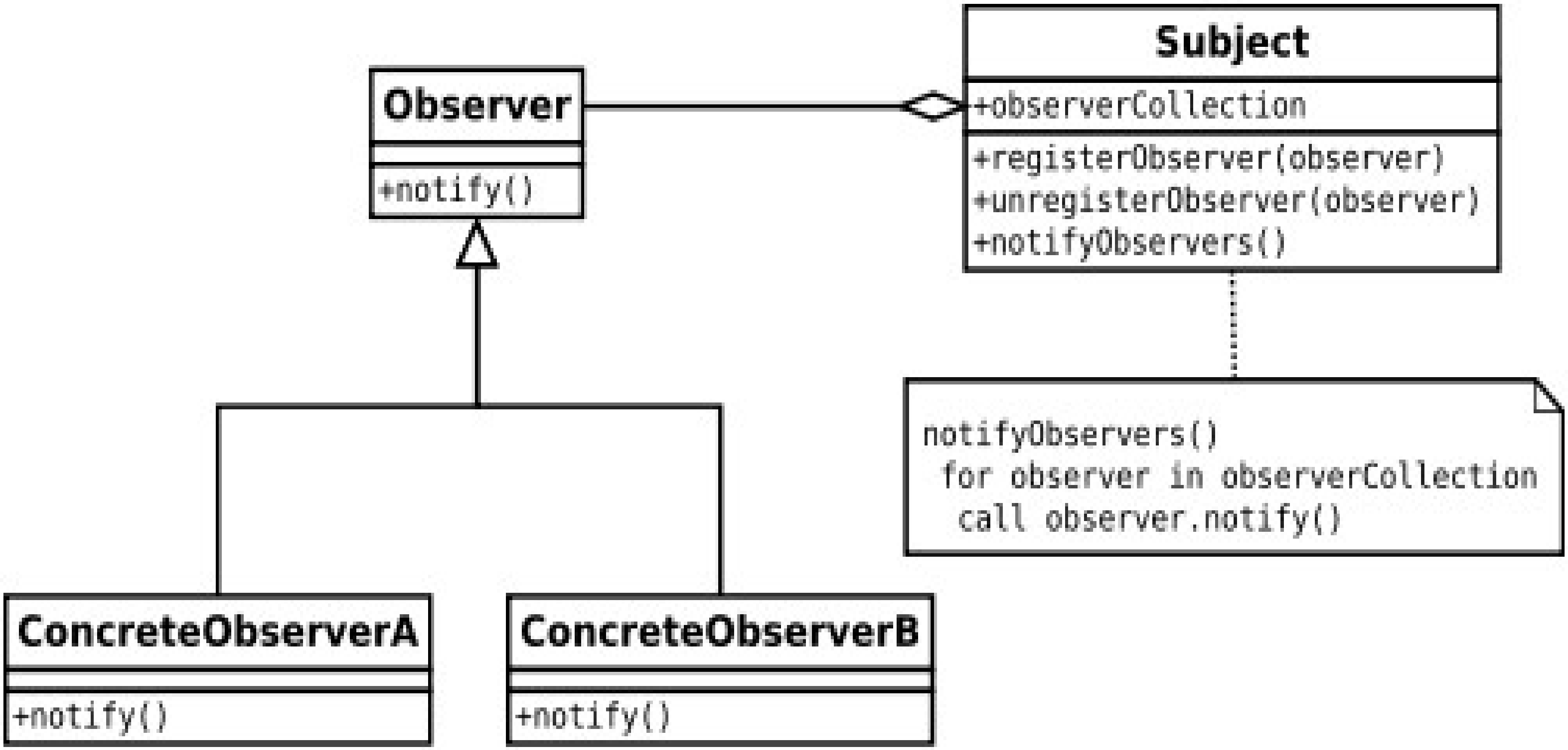
- If you(user of data/observer) are interested in the magazine you subscribe(register), and if a new edition is published it gets delivered to you.

- If you unsubscribe(unregister) you stop getting new editions.

- Publisher doesn’t know who you are and how you use the magazine, it just delivers it to you because you are a subscriber(loose coupling).

- Definition:
- The Observer Pattern defines a one to many dependency between objects so that one object changes state, all of its dependents are notified and updated automatically.
- Explanation:
- One to many dependency is between Subject(One) and Observer(Many).
- There is dependency as Observers themselves don’t have access to data.
- They are dependent on Subject to provide them data.

Class diagram:


Observer
+notify()
Subject
+observerCollection
+registerObserver(observer)
+unregisterObserver(observer)
+notifyObservers()
notifyObservers()
for observer in observerCollection
call observer.notify()
ConcreteObserverA
+notify()
ConcreteObserverB
+notify()

- Here Observer and Subject are interfaces(can be any abstract super type not necessarily java interface).
- All observers who need the data need to implement observer interface.
- notify() method in observer interface defines the action to be taken when the subject provides it data.
- The subject maintains an observerCollection which is simply the list of currently registered(subscribed) observers.
- registerObserver(observer) and unregisterObserver(observer) are methods to add and remove observers respectively.
- notifyObservers() is called when the data is changed and the observers need to be supplied with new data.

- Advantages:
- Provides a loosely coupled design between objects that interact.
- Loosely coupled objects are flexible with changing requirements.
- Here loose coupling means that the interacting objects should have less information about each other.
- Observer pattern provides this loose coupling as:
- Subject only knows that observer implement Observer interface. Nothing more.
- There is no need to modify Subject to add or remove observers.
- We can reuse subject and observer classes independently of each other.

- Disadvantages:

- Memory leaks caused by Lapsed listener problem because of explicit register and unregistering of observers.

- When to use this pattern?

- You should consider using this pattern in your application when multiple objects are dependent on the state of one object as it provides a neat and well tested design for the same.

Real Life Uses:

- It is heavily used in GUI toolkits and event listener. In java the button(subject) and onClickListener(observer) are modelled with observer pattern.
- Social media, RSS feeds, email subscription in which you have the option to follow or subscribe and you receive latest notification.
- All users of an app on play store gets notified if there is an update.

Observer Pattern –Introduction

- In Set 1, we discussed below problem, a solution for the problem without Observer pattern and problems with the solution.

- *Suppose we are building a cricket app that notifies viewers about the information such as current score, run rate etc.*

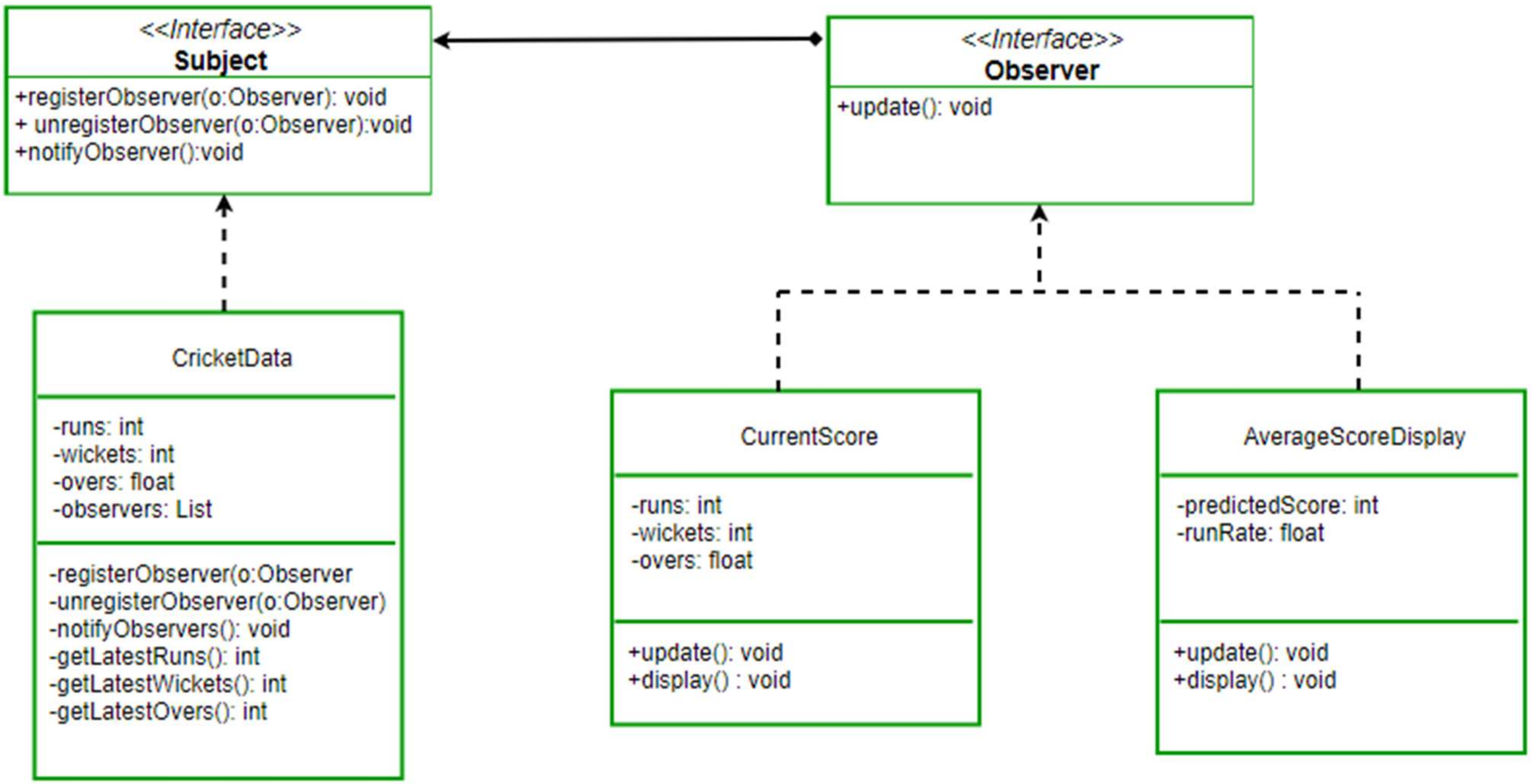
- *Suppose we have made two display elements CurrentScoreDisplay and AverageScoreDisplay. CricketData has all the data (runs, bowls etc.) and whenever data changes the display elements are notified with new data and they display the latest data accordingly*

- Applying Observer pattern to above problem:

- Let us see how we can improve the design of our application using observer pattern.

- If we observe the flow of data we can easily see that the CricketData and display elements follow subject-observers relationship.

New Class Diagram:


<<Interface>>
Subject
+registerObserver(o:Observer): void
+ unregisterObserver(o:Observer):void
+notifyObserver():void
<<Interface>>
Observer
+update(): void
CricketData
-runs: int
-wickets: int
-overs: float
-observers: List
-registerObserver(o:Observer
-unregisterObserver(o:Observer)
-notifyObservers(): void
-getLatestRuns(): int
-getLatestWickets(): int
-getLatestOvers(): int
CurrentScore
-runs: int
-wickets: int
-overs: float
+update(): void
+display() : void
AverageScoreDisplay
-predictedScore: int
-runRate: float
+update(): void
+display() : void

```
// Java program to demonstrate working of observer pattern
import java.util.ArrayList;
import java.util.Iterator;

// Implemented by Cricket data to communicate with observers
interface Subject
{
   public void registerObserver(Observer o);
   public void unregisterObserver(Observer o);
   public void notifyObservers();
}

class CricketData implements Subject
{
   int runs;
   int wickets;
   float overs;
   ArrayList<Observer> observerList;

   public CricketData() {
      observerList = new ArrayList<Observer>();
   }
```

```
@Override
 public void registerObserver(Observer o) {
     observerList.add(o);
 }

 @Override
 public void unregisterObserver(Observer o) {
     observerList.remove(observerList.indexOf(o));
 }

 @Override
 public void notifyObservers()
 {
     for (Iterator<Observer> it =
           observerList.iterator(); it.hasNext();)
     {
         Observer o = it.next();
         o.update(runs,wickets,overs);
     }
 }
```

```
// get latest runs from stadium
   private int getLatestRuns()
   {
      // return 90 for simplicity
      return 90;
   }

   // get latest wickets from stadium
   private int getLatestWickets()
   {
      // return 2 for simplicity
      return 2;
   }

   // get latest overs from stadium
   private float getLatestOvers()
   {
      // return 90 for simplicity
      return (float)10.2;
   }
```

```
// This method is used update displays when data changes

    public void dataChanged()
    {
        //get latest data
        runs = getLatestRuns();
        wickets = getLatestWickets();
        overs = getLatestOvers();

        notifyObservers();
    }
}
```

```
// This interface is implemented by all those classes that are to be updated whenever there is an update from CricketData
interface Observer
{
   public void update(int runs, int wickets,
                  float overs);
}

class AverageScoreDisplay implements Observer
{
   private float runRate;
   private int predictedScore;

   public void update(int runs, int wickets, float overs)
   {
      this.runRate =(float)runs/overs;
      this.predictedScore = (int)(this.runRate * 50);
      display();
   }

   public void display()
   {
      System.out.println("\nAverage Score Display: \n"
                     + "Run Rate: " + runRate +
                     "\nPredictedScore: " +
                     predictedScore);
   }
}
```

```
class CurrentScoreDisplay implements Observer
{
    private int runs, wickets;
    private float overs;

    public void update(int runs, int wickets, float overs)
    {
        this.runs = runs;
        this.wickets = wickets;
        this.overs = overs;
        display();
    }

    public void display()
    {
        System.out.println("\nCurrent Score Display:\n"
                           + "Runs: " + runs +
                           "\nWickets:" + wickets +
                           "\nOvers: " + overs );
    }
}
```

```
// Driver Class
class Main
{
   public static void main(String args[])
   {
      // create objects for testing
      AverageScoreDisplay averageScoreDisplay =
                     new AverageScoreDisplay();
      CurrentScoreDisplay currentScoreDisplay =
                     new CurrentScoreDisplay();

      // pass the displays to Cricket data
      CricketData cricketData = new CricketData();
// register display elements
        cricketData.registerObserver(averageScoreDisplay);
        cricketData.registerObserver(currentScoreDisplay);

        // in real app you would have some logic to call this function when data changes
        cricketData.dataChanged();

        //remove an observer
        cricketData.unregisterObserver(averageScoreDisplay);

        // now only currentScoreDisplay gets the notification
        cricketData.dataChanged();
    }
}
```

Output:

```
Average Score Display:
Run Rate: 8.823529
 PredictedScore: 441

Current Score Display:
Runs: 90
Wickets:2
Overs: 10.2


Current Score Display:
Runs: 90
Wickets:2
Overs: 10.2
```

Note: Now we can add/delete as many observers without changing the subject.

The Decorator Design Pattern

The Decorator Design Pattern

- Using the decorator design pattern is fairly simple.

- You can have a base class with methods and properties that are present when you make a new object with the class.

- Now say you have some instances of the class that need methods or properties that didn't come from the base class.

- You can add those extra methods and properties to the base class, but that could mess up your other instances.

- You could even make sub-classes to hold specific methods and properties you need that you can't put in your base class.

- Here's an example of a customer class:

```
class Customer {
  constructor(balance=20) {
    this.balance = balance
    this.foodItems = []
  }
  buy(food) {
    if (food.price) < this.balance {
      console.log('you should get it')
      this.balance -= food.price
      this.foodItems.push(food)
    }
    else {
      console.log('maybe you should get something else')
    }
  }
}
module.exports = Customer
```

```
class ExquisiteSandwich {
  constructor(baseSandwich) {
    this.type = `Exquisite ${baseSandwich.type}`
    this.price = baseSandwich.price + 10.75
  }

  order() {
    console.log(`You ordered an ${this.type} sandwich. It's got everything you need to be happy for days.`)
  }
}

module.exports = { Sandwich, DeluxeSandwich, ExquisiteSandwich }
```

- This sandwich class is where the decorator pattern is used.

- We have a *Sandwich* base class that sets the rules for what happens when a regular sandwich is ordered.

- Customers might want to upgrade sandwiches and that just means an ingredient and price change.

- You just wanted to add the functionality to increase the price and update the type of sandwich for the *DeluxeSandwich* without changing how it's ordered.

- Although you might need a different order method for an *ExquisiteSandwich* because there is a drastic change in the quality of ingredients.

- The decorator pattern lets you dynamically change the base class without affecting it or any other classes.

- You don't have to worry about implementing functions you don't know, like with interfaces, and you don't have to include properties you won't use in every class.

- Now if we'll go over an example where this class is instantiated as if a customer was placing a sandwich order.

- const { Sandwich, DeluxeSandwich, ExquisiteSandwich } = require('./Sandwich')
- const Customer = require('./Customer')

- const cust1 = new Customer(57)

- const turkeySandwich = new Sandwich('Turkey', 6.49)
- const bltSandwich = new Sandwich('BLT', 7.55)

- const deluxeBltSandwich = new DeluxeSandwich(bltSandwich)
- const exquisiteTurkeySandwich = new ExquisiteSandwich(turkeySandwich)

- cust1.buy(turkeySandwich)
- cust1.buy(bltSandwich)

BITS Pilani
Pilani Campus

Introduction to Design Patterns(GOF)

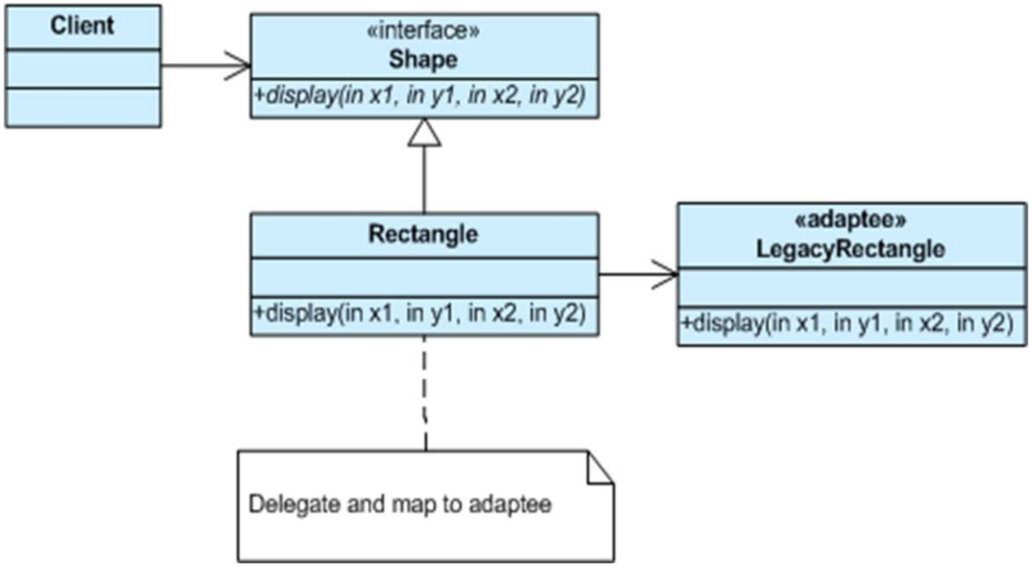
By Prof A R Rahman

The adapter design pattern

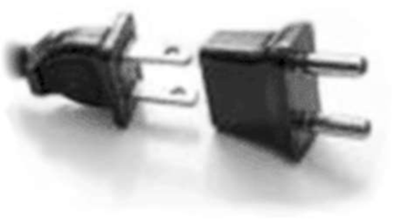
The adapter design pattern

- When two software components (e.g., legacy code and new development or a COTS) cannot interface
- *Adapter* changes interface of one so the other can use it
    - *Adapter* fills the gap b/w two interfaces
    - No changes needed for either

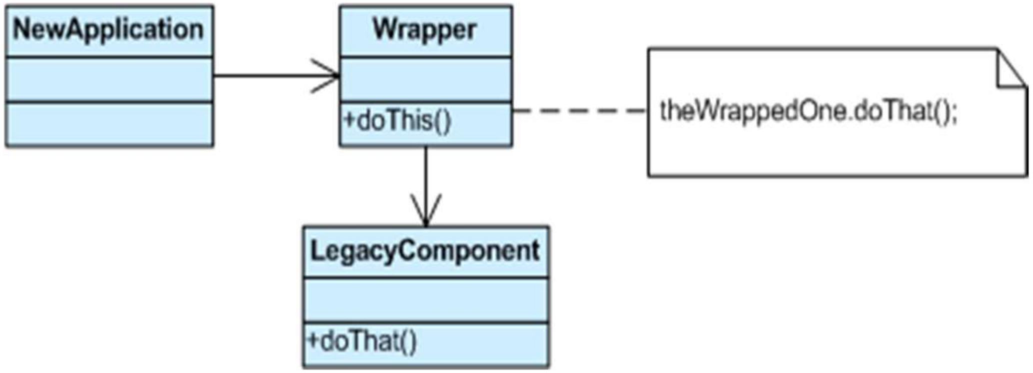
Yet another design pattern: adapter


Client
«interface»
Shape
+display(in x1, in y1, in x2, in y2)
Rectangle
+display(in x1, in y1, in x2, in y2)
«adaptee»
LegacyRectangle
+display(in x1, in y1, in x2, in y2)
Delegate and map to adaptee

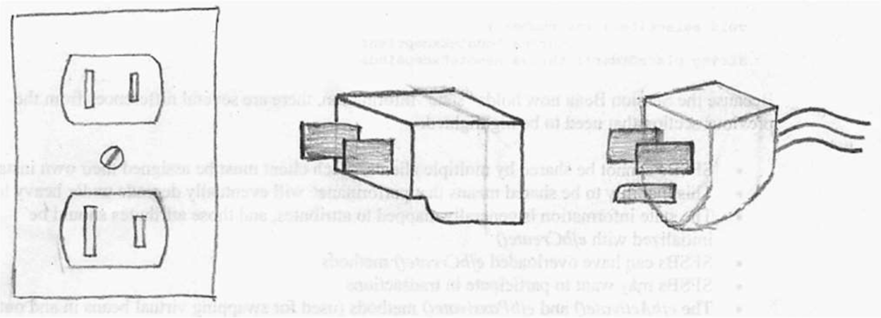
```
class NewTime
{
public:
int GetTime() {
    return otime.get_time() * 100;
   }
private:
   OriginalTime otime;
};
```

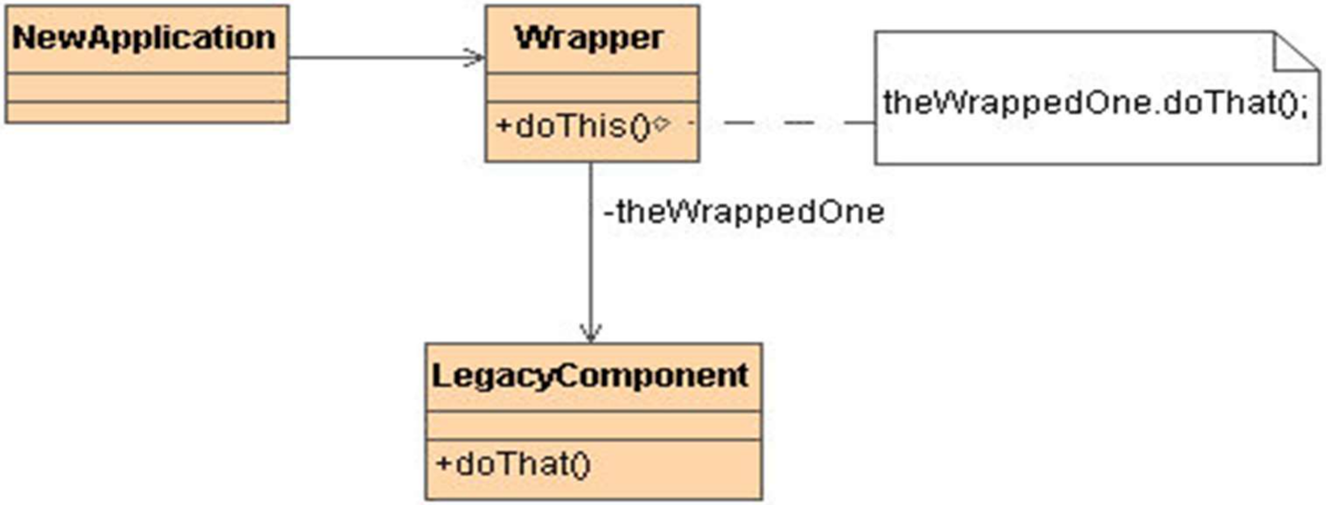


An alternate: a *wrapper*


NewApplication
Wrapper
+doThis()
theWrappedOne.doThat();
LegacyComponent
+doThat()

Yet another design pattern: adapter





NewApplication

Wrapper

+doThis()

theWrappedOne.doThat();

-theWrappedOne

LegacyComponent

+doThat()

- This pattern is easy to understand as the real world is full of adapters.
- For example consider a USB to Ethernet adapter.
- We need this when we have an Ethernet interface on one end and USB on the other.
- Since they are incompatible with each other. we use an adapter that converts one to other.
- This example is pretty analogous to Object Oriented Adapters.
- In design, adapters are used when we have a class (Client) expecting some type of object and we have an object (Adaptee) offering the same features but exposing a different interface.

To use an adapter:

The client makes a request to the adapter by calling a method on it using the target interface.

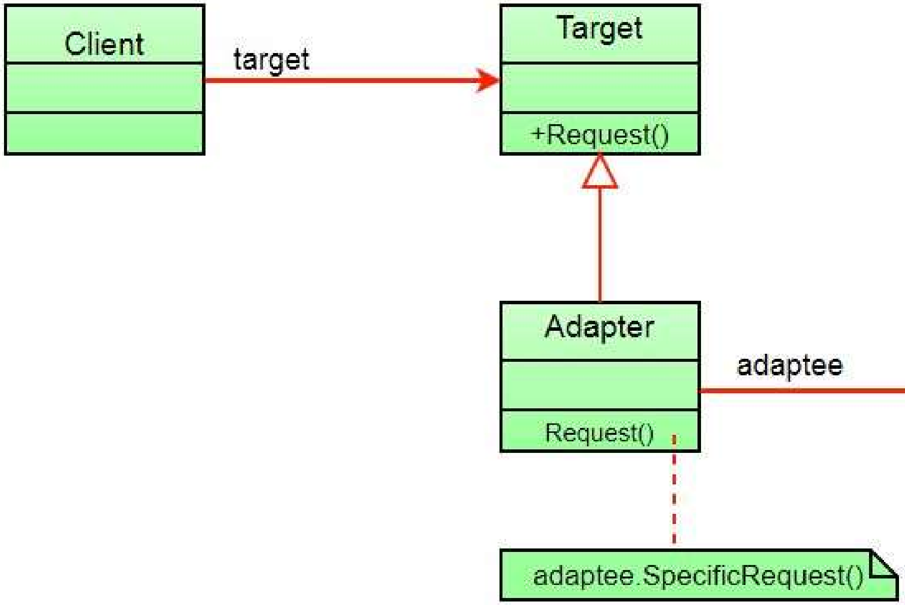
The adapter translates that request on the adaptee using the adaptee interface.

Client receive the results of the call and is unaware of adapter’s presence.

Definition:

The adapter pattern convert the interface of a class into another interface clients expect.

Adapter lets classes work together that couldn’t otherwise because of incompatible interfaces.


Client
target
Target
+Request()
Adapter
Request()
adaptee
Adapter
+SpecificRequest()
adaptee.SpecificRequest()

- The client sees only the target interface and not the adapter.
- The adapter implements the target interface.
- Adapter delegates all requests to Adaptee.

Example:

Suppose you have a Bird class with fly() , and makeSound()methods.

And also a ToyDuck class with squeak() method.

Let’s assume that you are short on ToyDuck objects and you would like to use Bird objects in their place.

Birds have some similar functionality but implement a different interface, so we can’t use them directly.

So we will use adapter pattern.

Here our client would be ToyDuck and adaptee would be Bird.

Below is Java implementation of it.

```
// Java implementation of Adapter pattern

interface Bird
{
   // birds implement Bird interface that allows them to fly and make sounds adaptee interface
   public void fly();
   public void makeSound();
}

class Sparrow implements Bird
{
   // a concrete implementation of bird
   public void fly()
   {
      System.out.println("Flying");
   }
   public void makeSound()
   {
      System.out.println("Chirp Chirp");
   }
}
```

```
interface ToyDuck
{
   // target interface
   // toyducks dont fly they just make  squeaking sound
   public void squeak();
}

class PlasticToyDuck implements ToyDuck
{
   public void squeak()
   {
      System.out.println("Squeak");
   }
}
```

```
class BirdAdapter implements ToyDuck
{
    // You need to implement the interface your  client expects to use.
    Bird bird;
    public BirdAdapter(Bird bird)
    {
        // we need reference to the object we are adapting
        this.bird = bird;
    }

    public void squeak()
    {
        // translate the methods appropriately
        bird.makeSound();
    }
}
```

```
class Main
{
    public static void main(String args[])
    {
        Sparrow sparrow = new Sparrow();
        ToyDuck toyDuck = new PlasticToyDuck();

        // Wrap a bird in a birdAdapter so that it behaves like toy duck
        ToyDuck birdAdapter = new BirdAdapter(sparrow);

        System.out.println("Sparrow...");
        sparrow.fly();
        sparrow.makeSound();

        System.out.println("ToyDuck...");
        toyDuck.squeak();

        // toy duck behaving like a bird
        System.out.println("BirdAdapter...");
        birdAdapter.squeak();
    }
}
```

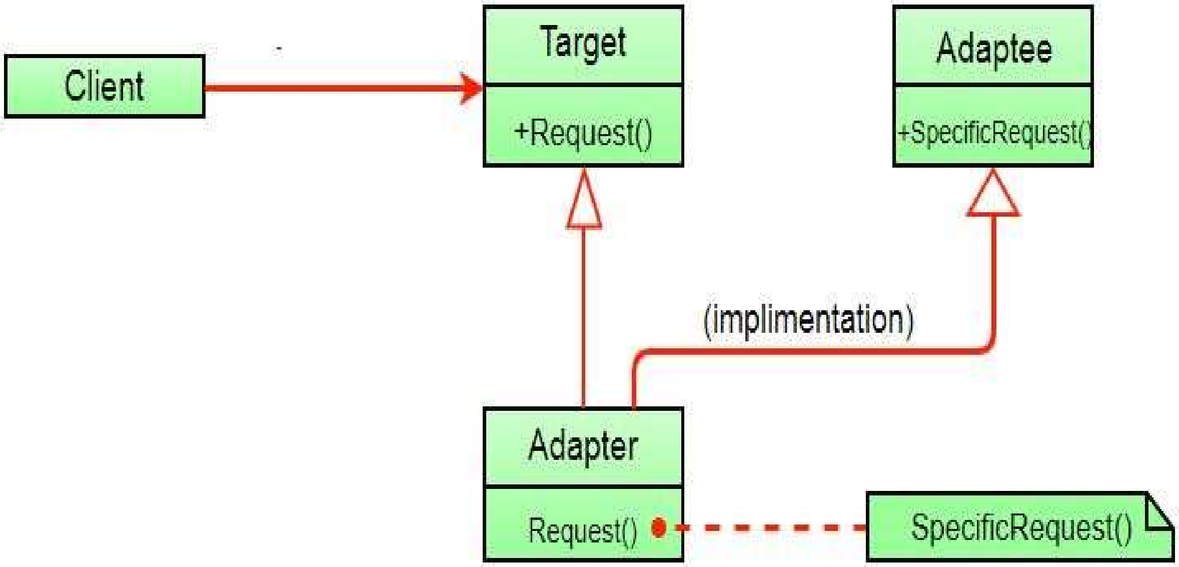
Output:

```
Sparrow...
Flying Chirp Chirp
ToyDuck...
Squeak
BirdAdapter...
Chirp Chirp
```

- Explanation :
- Suppose we have a bird that can makeSound(), and we have a plastic toy duck that can squeak().
- Now suppose our client changes the requirement and he wants the toyDuck to makeSound than ?
- Simple solution is that we will just change the implementation class to the new adapter class and tell the client to pass the instance of the bird(which wants to squeak()) to that class.
- Before : ToyDuck toyDuck = new PlasticToyDuck();
- After : ToyDuck toyDuck = new BirdAdapter(sparrow);
- You can see that by changing just one line the toyDuck can now do Chirp Chirp !!

- Object Adapter Vs Class Adapter
- The adapter pattern we have implemented above is called Object Adapter Pattern because the adapter holds an instance of adaptee.
- There is also another type called Class Adapter Pattern which use inheritance instead of composition but you require multiple inheritance to implement it.

- Class diagram of Class Adapter Pattern:


Client
Target
+Request()
Adaptee
+SpecificRequest()
(implimentation)
Adapter
Request()
SpecificRequest()

- Here instead of having an adaptee object inside adapter (composition) to make use of its functionality adapter inherits the adaptee.

- Since multiple inheritance is not supported by many languages including java and is associated with many problems we have not shown implementation using class adapter pattern.

- Advantages:

- Helps achieve reusability and flexibility.

- Client class is not complicated by having to use a different interface and can use polymorphism to swap between different implementations of adapters.

- Disadvantages:

- All requests are forwarded, so there is a slight increase in the overhead.

- Sometimes many adaptations are required along an adapter chain to reach the type which is required.

```
#include <iostream.h>
typedef int Coordinate;
typedef int Dimension;

//////////////////////// Desired interface ////////////////////////
class Rectangle { public: virtual void draw() = 0; };

//////////////////////// Legacy component ////////////////////////
class LegacyRectangle {
public:
    LegacyRectangle( Coordinate x1, Coordinate y1, Coordinate x2, Coordinate y2 )
                { x1_ = x1; y1_ = y1; x2_ = x2; y2_ = y2;
                  cout << "LegacyRectangle: create. (" << x1_ << "," << y1_ << ") => (" << x2_<<   ",” << y2_ << ")" << endl; }
    void oldDraw()
                { cout << "LegacyRectangle: oldDraw. (" << x1_ << "," << y1_ << ") => (" << x2_<< "," << y2_ << ")" << endl; }
private:
    Coordinate x1_; Coordinate y1_; Coordinate x2_; Coordinate y2_;
};
```

```
////////////////// Adapter wrapper //////////////////
class RectangleAdapter : public Rectangle, private LegacyRectangle {
public:
   RectangleAdapter( Coordinate x, Coordinate y, Dimension w,
   Dimension h ) : LegacyRectangle( x, y, x+w, y+h )
      { cout << "RectangleAdapter: create. (" << x << "," << y << "),
   width = " << w << ", height = " << h << endl; }
   virtual void draw()
            { cout << "RectangleAdapter: draw." << endl;
              oldDraw(); }
};
   void main()
   { Rectangle* r = new RectangleAdapter( 120, 200, 60, 40 );
       r->draw(); }

// LegacyRectangle: create. (120,200) => (180,240)
// RectangleAdapter: create. (120,200), width = 60, height = 40
// RectangleAdapter: draw.
// LegacyRectangle: oldDraw. (120,200) => (180,240)
```

- Adapter method is a Structural Design Pattern which helps us in making the incompatible objects adaptable to each other.
- The Adapter method is one of the easiest methods to understand because we have a lot of real-life examples that show the analogy with it.
- The main purpose of this method is to create a bridge between two incompatible interfaces.
- This method provides a different interface for a class.
- We can more easily understand the concept by thinking about the Cable Adapter that allows us to charge a phone somewhere that has outlets in different shapes.
- Using this idea, we can integrate the classes that couldn’t be integrated due to interface incompatibility.

- Problem without using the Adapter Method

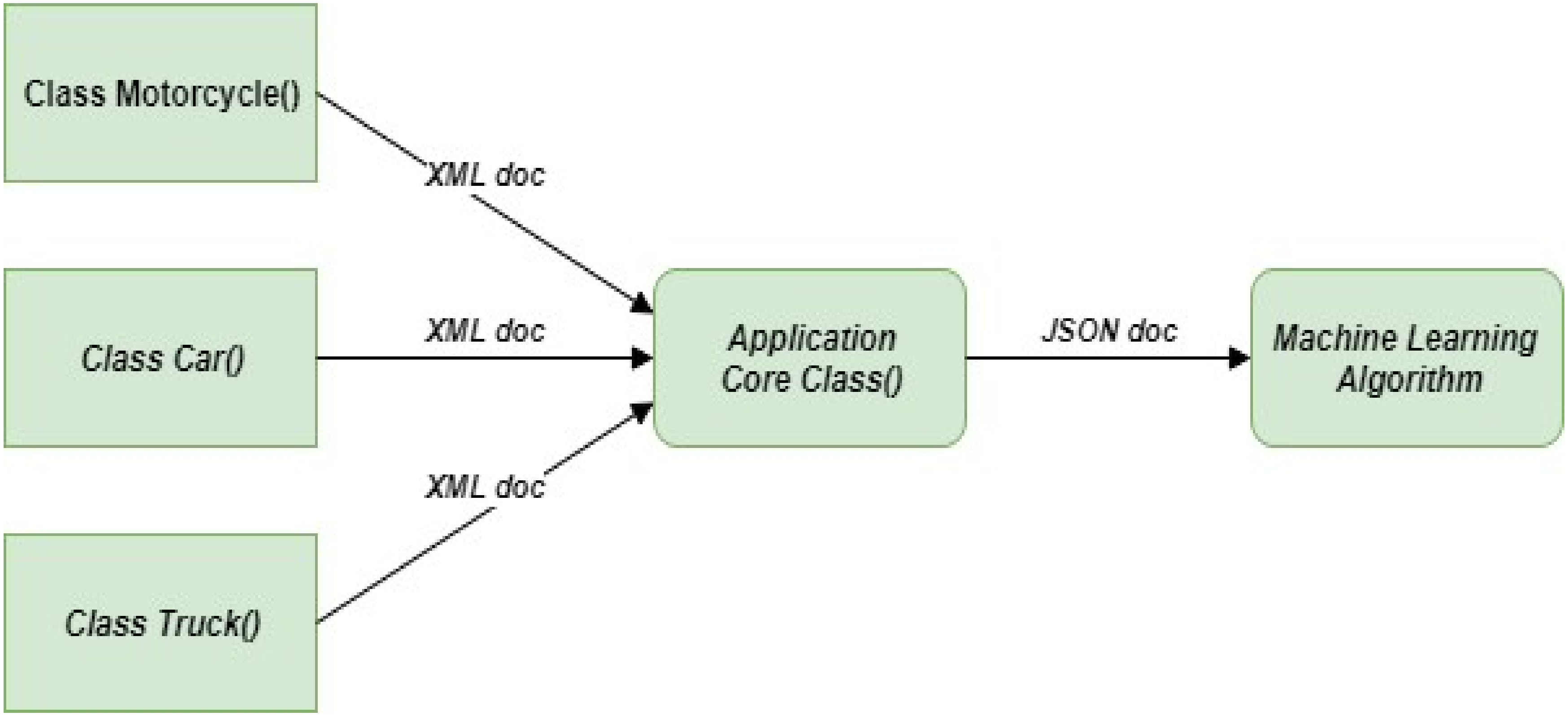
- Imagine you are creating an application that shows the data about all different types of vehicles present.

- It takes the data from APIs of different vehicle organizations in XML format and then displays the information.

- But suppose at some time you want to upgrade your application with a Machine Learning algorithms that work beautifully on the data and fetch the important data only.

- But there is a problem, it takes data in JSON format only. It will be a really poor approach to make changes in Machine Learning Algorithm so that it will take data in XML format.

Problem-Adapter-Method


Class Motorcycle()
Class Car()
Class Truck()
XML doc
XML doc
XML doc
Application Core Class()
JSON doc
Machine Learning Algorithm

- Solution using Adapter Method

- For solving the problem we defined above, We can use Adapter Method that helps by creating an Adapter object.

- To use an adapter in our code:

- Client should make a request to the adapter by calling a method on it using the target interface.

- Using the Adaptee interface, the Adapter should translate that request on the adaptee.

- Result of the call is received the client and he/she is unaware of the presence of the Adapter's presence.

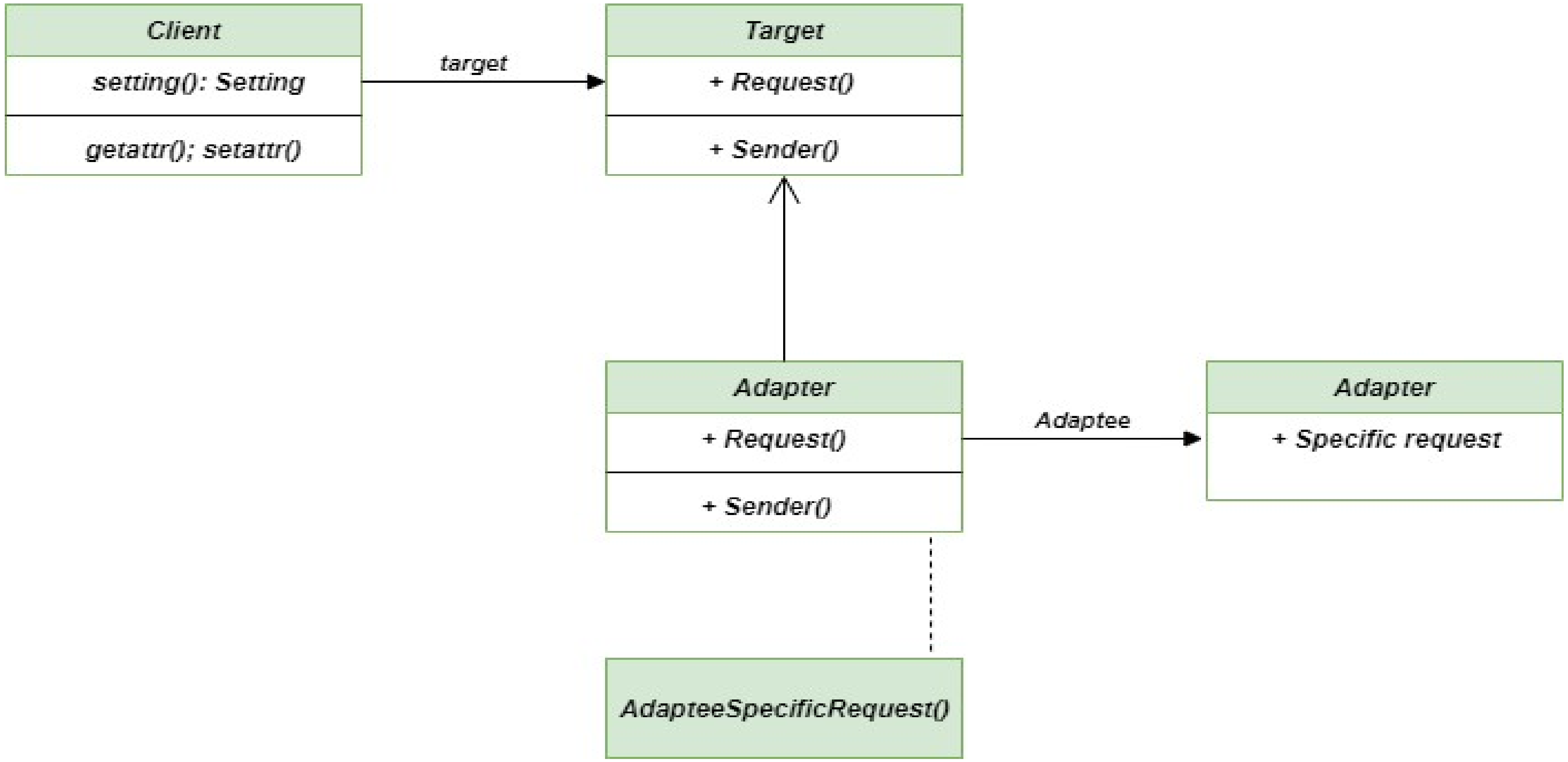
```
•  # Dog - Cycle
•  # human - Truck
•  # car - Car

•  class MotorCycle:

•      """Class for MotorCycle"""

•      def __init__(self):
•          self.name = "MotorCycle"

•      def TwoWheeler(self):
•          return "TwoWheeler"


•      def EightWheeler(self):
•          return "EightWheeler"
•   class Truck:

•           """Class for Truck"""

•           def __init__(self):
•                   self.name = "Truck"

•           def EightWheeler(self):
•                   return "EightWheeler"


•   class Car:

•           """Class for Car"""

•           def __init__(self):
•                   self.name = "Car“

            def FourWheeler(self):
                    return "FourWheeler"
```

```
class Adapter:
    """
    Adapts an object by replacing methods.
    Usage:
    motorCycle = MotorCycle()
    motorCycle = Adapter(motorCycle, wheels = motorCycle.TwoWheeler)
    """

    def __init__(self, obj, **adapted_methods):
        """We set the adapted methods in the object's dict"""
        self.obj = obj
        self.__dict__.update(adapted_methods)

    def __getattr__(self, attr):
        """All non-adapted calls are passed to the object"""
        return getattr(self.obj, attr)

    def original_dict(self):
        """Print original object dict"""
        return self.obj.__dict__
```

```
""" main method """
if __name__ == "__main__":

    """list to store objects"""
    objects = []

    motorCycle = MotorCycle()
    objects.append(Adapter(motorCycle, wheels = motorCycle.TwoWheeler))

    truck = Truck()
    objects.append(Adapter(truck, wheels = truck.EightWheeler))

    car = Car()
    objects.append(Adapter(car, wheels = car.FourWheeler))

    for obj in objects:
        print("A {0} is a {1} vehicle".format(obj.name, obj.wheels()))
```

Class Diagram

Class diagram for the Adapter method which is a type of Structural Design pattern:


Client
setting(): Setting
getattr(); setattr()
target
Target
+ Request()
+ Sender()
Adapter
+ Request()
+ Sender()
Adaptee
Adapter
+ Specific request
AdapteeSpecificRequest()

Advantages

Principle of Single Responsibility: We can achieve the principle of Single responsibility with Adapter Method because here we can separate the concrete code from the primary logic of the client.

Flexibility: Adapter Method helps in achieving the flexibility and reusability of the code.

Less complicated class: Our client class is not complicated by having to use a different interface and can use polymorphism to swap between different implementations of adapters.

Open/Closed principle: We can introduce the new adapter classes into the code without violating the Open/Closed principle.

Disadvantages

Complexity of the Code: As we have introduced the set of new classes, objects and interfaces, the complexity of or code definitely rises.

Adaptability: Most of the times, we require many adaptations with the adaptee chain to reach the compatibility what we want.

Applicability

To make classes and interfaces compatible : Adapter method is always used when we are in need to make certain classes compatible to communicate.

Relatable to Inheritance: When we want to reuse some piece of code i.e., classes and interfaces that lack some functionalities, it can be done using the Adapter Method.

Factory method design pattern

- The factory method is a creational design pattern, i.e., related to object creation.
- In the Factory pattern, we create objects without exposing the creation logic to the client and the client uses the same common interface to create a new type of object.
- The idea is to use a static member-function (static factory method) that creates & returns instances, hiding the details of class modules from the user.
- A factory pattern is one of the core design principles to create an object, allowing clients to create objects of a library(explained below) in a way such that it doesn't have a tight coupling with the class hierarchy of the library.
- *What is meant when we talk about* libraries *and clients?*
- A library is something that is provided by some third party that exposes some public APIs and clients make calls to those public APIs to complete their tasks.
- A very simple example can be different kinds of Views provided by Android OS.

```
// A design without factory pattern
#include <iostream>
using namespace std;

// Library classes
class Vehicle {
public:
    virtual void printVehicle() = 0;
};
class TwoWheeler : public Vehicle {
public:
    void printVehicle()
    {
        cout << "I am two wheeler" << endl;
    }
};
class FourWheeler : public Vehicle {
public:
    void printVehicle()
    {
        cout << "I am four wheeler" << endl;
    }
};
```

```
// Client (or user) class
class Client {
public:
        Client(int type)
        {

                // Client explicitly creates classes according to
                // type
                if (type == 1)
                        pVehicle = new TwoWheeler();
                else if (type == 2)
                        pVehicle = new FourWheeler();
                else
                        pVehicle = NULL;
        }

~Client()
        {
                if (pVehicle) {
                        delete pVehicle;
                        pVehicle = NULL;
                }
        }

        Vehicle* getVehicle() { return pVehicle; }

private:
        Vehicle* pVehicle;
};

// Driver program
int main()
{
        Client* pClient = new Client(1);
        Vehicle* pVehicle = pClient->getVehicle();
        pVehicle->printVehicle();
        return 0;
}
```

Output

```
I am two wheeler
```

- *What* are *the problems with* the *above design?*
- As you must have observed in the above example, the Client creates objects of either TwoWheeler or FourWheeler based on some input during constructing of its object.
- Say, the library introduces a new class ThreeWheeler to incorporate three-wheeler vehicles also.
- What would happen? The client will end up chaining a new else if in the conditional ladder to create objects of ThreeWheeler.
- Which in turn will need the Client to be recompiled.
- So, each time a new change is made at the library side, the Client would need to make some corresponding changes at its end and recompile the code.
- Sounds bad? This is a very bad practice of design.

How do avoid the problem?

The answer is, to create a static (or factory) method. Let us see the below code.

```
// C++ program to demonstrate factory method design pattern
#include <iostream>
using namespace std;

enum VehicleType {
   VT_TwoWheeler,    VT_ThreeWheeler,    VT_FourWheeler
};

// Library classes
class Vehicle {
public:
   virtual void printVehicle() = 0;
   static Vehicle* Create(VehicleType type);
   virtual ~Vehicle(){}
};
class TwoWheeler : public Vehicle {
public:
   void printVehicle() {
      cout << "I am two wheeler" << endl;
   }
};
```

```
class ThreeWheeler : public Vehicle {
public:
    void printVehicle() {
        cout << "I am three wheeler" << endl;
    }
};
class FourWheeler : public Vehicle {
    public:
    void printVehicle() {
        cout << "I am four wheeler" << endl;
    }
};
```

```
// Factory method to create objects of different types.

// Change is required only in this function to create a new object type

Vehicle* Vehicle::Create(VehicleType type)
{

   if (type == VT_TwoWheeler)
      return new TwoWheeler();

   else if (type == VT_ThreeWheeler)
      return new ThreeWheeler();

   else if (type == VT_FourWheeler)
      return new FourWheeler();

   else return NULL;
}
```

```
// Client class
class Client {
public:

   // Client doesn't explicitly create objects but passes type to factory method "Create()"
   Client()
   {
      VehicleType type = VT_ThreeWheeler;
      pVehicle = Vehicle::Create(type);
   }
   ~Client() {
      if (pVehicle) {
         delete pVehicle;
         pVehicle = NULL;
      }
   }
   Vehicle* getVehicle()  {
      return pVehicle;
   }

private:
   Vehicle *pVehicle;
};
```

```
// Driver program
int main() {
   Client *pClient = new Client();
   Vehicle * pVehicle = pClient->getVehicle();
   pVehicle->printVehicle();
     delete pClient;
   return 0;
}
```

Output

```
I am three wheeler
```

In the above example, we have totally decoupled the selection of types for object creation from the Client.

The library is now responsible to decide which object type to create based on an input.

The client just needs to make calls to the library’s factory Create method and pass the type it wants without worrying about the actual implementation of the creation of objects.

Other examples of the Factory Method:

Say, in a ‘Drawing’ system, depending on the user’s input, different pictures like squares, rectangles, the circle can be drawn.

Here we can use the factory method to create instances depending on the user’s input. For adding a new type of shape, no need to change the client’s code.

Another example: On the travel site, we can book train tickets as well as bus tickets and flight tickets.

In this case, the user can give his travel type as ‘bus’, ‘train’, or ‘flight’.

Here we have an abstract class ‘AnyTravel’ with a static member function ‘GetObject’ which depending on the user’s travel type, will create & return an object of ‘BusTravel’ or ‘ TrainTravel’.

‘BusTravel’ or ‘ TrainTravel’ have common functions like passenger name, Origin, and destination parameters.

Factory Method –Design Patterns in Python

Factory Method is a Creational Design Pattern that allows an interface or a class to create an object, but lets subclasses decide which class or object to instantiate.

Using the Factory method, we have the best ways to create an object.

Here, objects are created without exposing the logic to the client, and for creating the new type of object, the client uses the same common interface.

Problems we face without Factory Method:

Imagine you are having your own startup which provides ridesharing in different parts of the country.

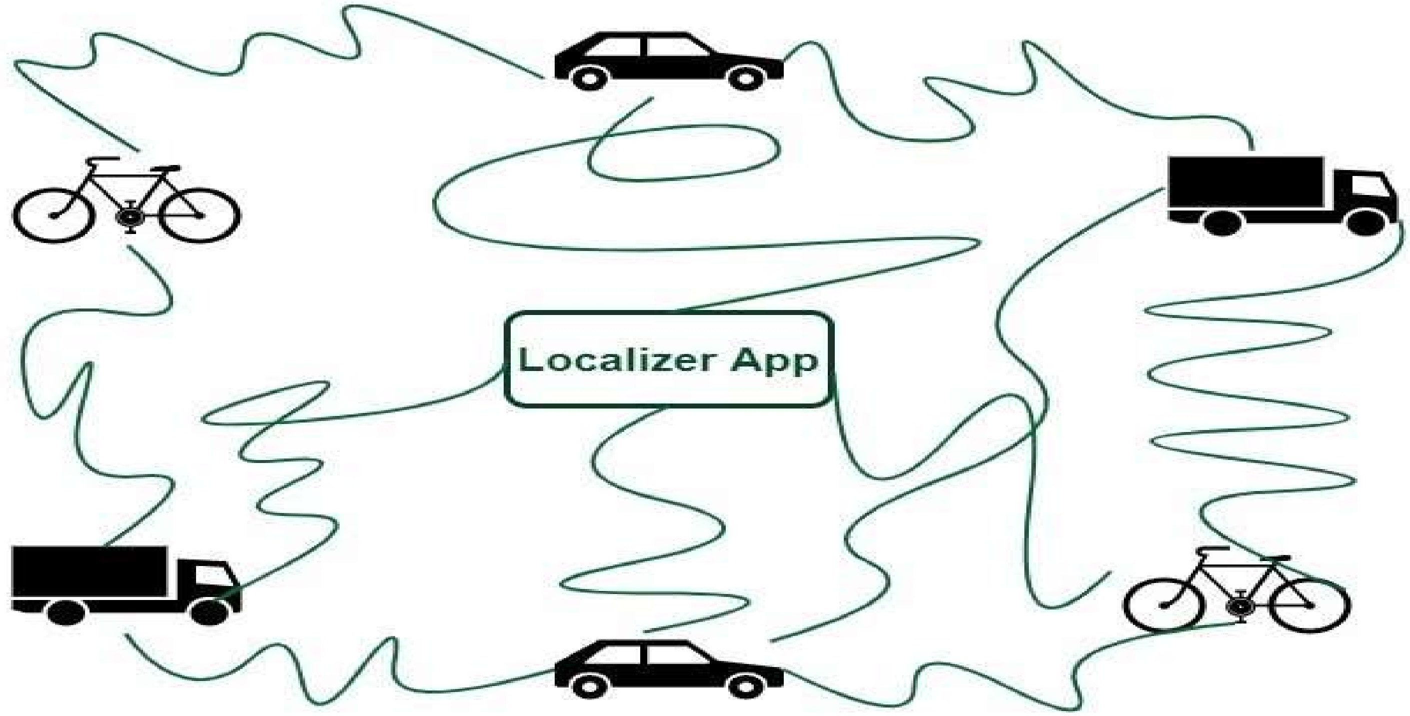
The initial version of the app only provides Two-Wheeler ridesharing but as time passes, your app becomes popular and now you want to add Three and Four-Wheeler ridesharing also.

It’s a piece of great news! but what about the software developers of your startup.

They have to change the whole code because now most part of the code is coupled with the Two-Wheeler class and developers have to make changes to the entire codebase.

After being done with all these changes, either the developers end with the messy code or with the resignation letter.

Localizer app



Diagrammatic representation of Problems without using Factory Method

Let’s understand the concept with one more example which is related to the translations and localization of the different languages.

Suppose we have created an app whose main purpose is to translate one language into another and currently our app works with 10 languages only.

Now our app has become widely popular among people but the demand has grown suddenly to include 5 more languages.

It’s a piece of great news! only for the owner not for the developers.

They have to change the whole code because now most part of the code is coupled with the existing languages only and that’s why developers have to make changes to the entire codebase which is really a difficult task to do.

Let’s look at the code for the problem which we may face without using the factory method.

Note: Following code is written without using the Factory method.

```
# Python Code for Object Oriented Concepts without using Factory method
class FrenchLocalizer:

        """ it simply returns the french version """

        def __init__(self):

                self.translations = {"car": "voiture", "bike": "bicyclette",
                                                        "cycle":"cyclette"}

        def localize(self, msg):

                """change the message using translations"""
                return self.translations.get(msg, msg)
```

```
class SpanishLocalizer:
        """it simply returns the spanish version"""

        def __init__(self):

                self.translations = {"car": "coche", "bike": "bicicleta",
                                                                        "cycle":"ciclo"}

        def localize(self, msg):

                """change the message using translations"""
                return self.translations.get(msg, msg)
```

```
class EnglishLocalizer:
	"""Simply return the same message"""

	def localize(self, msg):
		return msg

if __name__ == "__main__":

	# main method to call others
	f = FrenchLocalizer()
	e = EnglishLocalizer()
	s = SpanishLocalizer()

	# list of strings
	message = ["car", "bike", "cycle"]

	for msg in message:
		print(f.localize(msg))
		print(e.localize(msg))
		print(s.localize(msg))
```

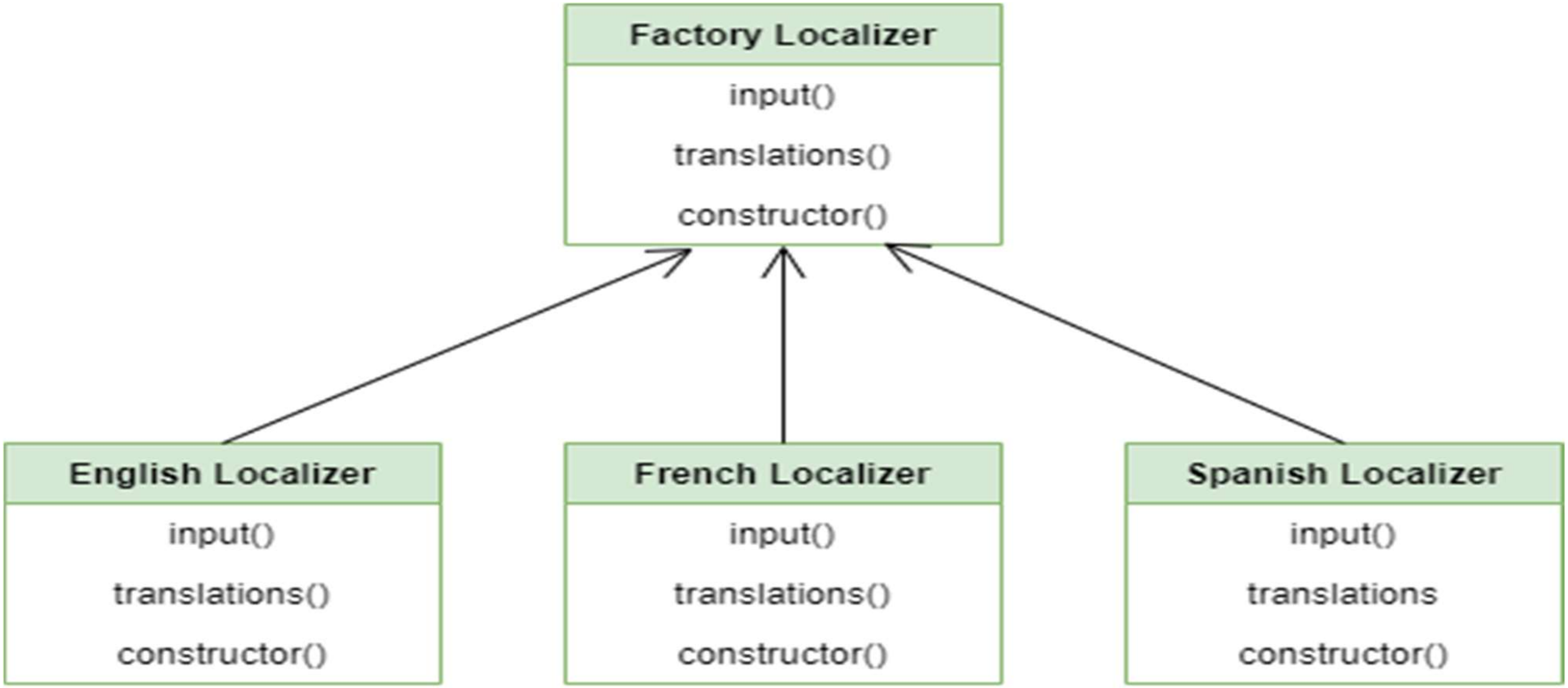
Solution by Factory Method:

Its solution is to replace the straightforward object construction calls with calls to the special factory method.

Actually, there will be no difference in the object creation but they are being called within the factory method.

For example Our Two_Wheeler, Three_Wheeler, and Four_wheeler classes should implement the ridesharing interface which will declare a method called a ride.

Each class will implement this method uniquely.


Factory Localizer
input()
translations()
constructor()
English Localizer
input()
translations()
constructor()
French Localizer
input()
translations()
constructor()
Spanish Localizer
input()
translations
constructor()

solution_factory_diagram

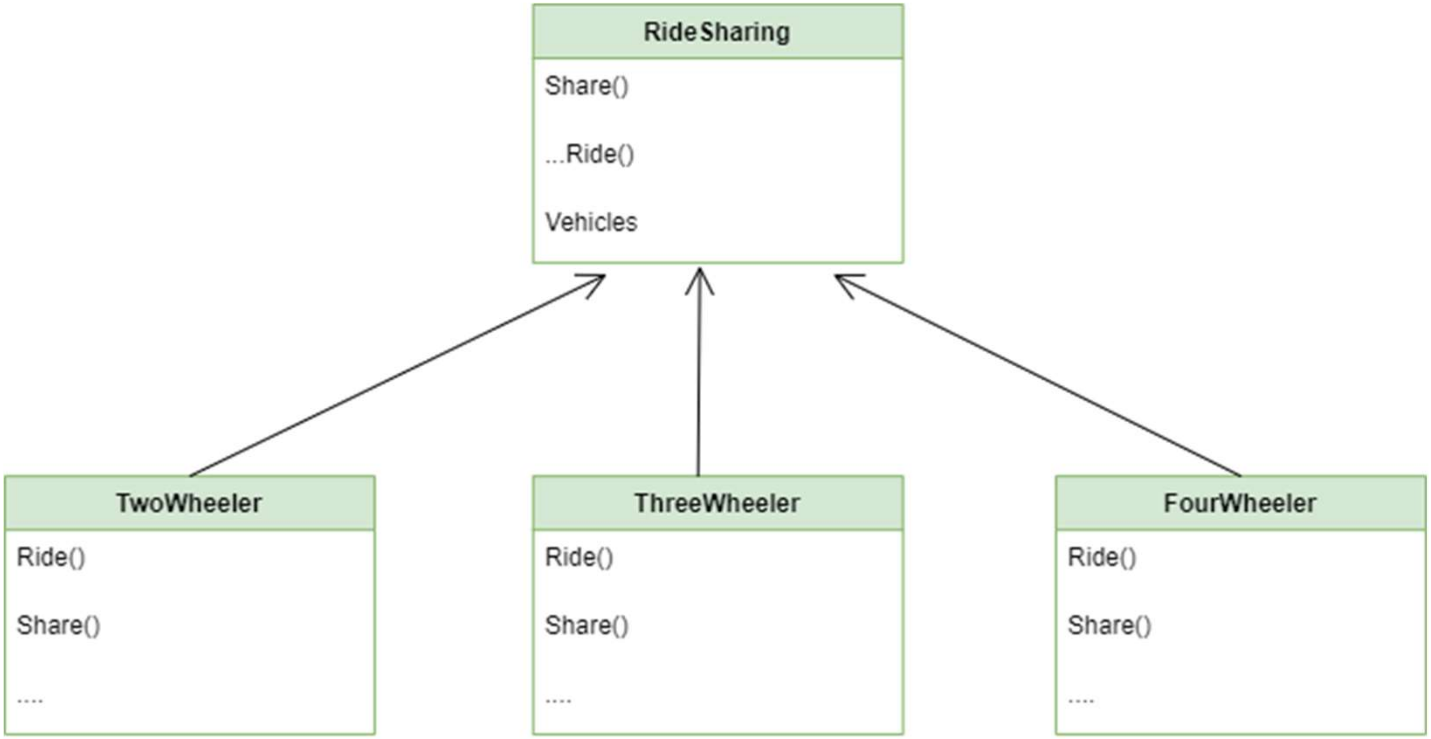
Now let us understand the factory method with the help of an example:

```
# Python Code for factory method it comes under the creational Design Pattern

class FrenchLocalizer:

        """ it simply returns the french version """

        def __init__(self):

                self.translations = {"car": "voiture", "bike": "bicyclette",
                                                                        "cycle":"cyclette"}

        def localize(self, msg):

                """change the message using translations"""
                return self.translations.get(msg, msg)
```

```
class SpanishLocalizer:
	"""it simply returns the spanish version"""

	def __init__(self):
		self.translations = {"car": "coche", "bike": "bicicleta", "cycle":"ciclo"}

	def localize(self, msg):

		"""change the message using translations"""
		return self.translations.get(msg, msg)

class EnglishLocalizer:
	"""Simply return the same message"""

	def localize(self, msg):
		return msg
```

```
def Factory(language ="English"):

        """Factory Method"""
        localizers = {
                "French": FrenchLocalizer,
                "English": EnglishLocalizer,
                "Spanish": SpanishLocalizer,
        }

        return localizers[language]()

if __name__ == "__main__":

        f = Factory("French")
        e = Factory("English")
        s = Factory("Spanish")

        message = ["car", "bike", "cycle"]

        for msg in message:
                print(f.localize(msg))
                print(e.localize(msg))
                print(s.localize(msg))
```

Class Diagram for Factory Method:
Let's look at the class diagram considering the example of ride-sharing.


RideSharing
Share()
...Ride()
Vehicles
TwoWheeler
Ride()
Share()
....
ThreeWheeler
Ride()
Share()
....
FourWheeler
Ride()
Share()
....

Factory_pattern_class_diagram

Advantages of using Factory method:

We can easily add new types of products without disturbing the existing client code.

Generally, tight coupling is being avoided between the products and the creator classes and objects.

Disadvantages of using Factory method:

To create a particular concrete product object, the client might have to sub-class the creator class.

You end up with a huge number of small files i.e, cluttering the files.

In a Graphics system, depending upon the user's input it can draw different shapes like rectangles, Square, Circle, etc.

But for the ease of both developers as well as the client, we can use the factory method to create the instance depending upon the user's input. Then we don't have to change the client code for adding a new shape.

On a Hotel booking site, we can book a slot for 1 room, 2 rooms, 3 rooms, etc.

Here user can input the number of rooms he wants to book. Using the factory method, we can create a factory class Any Rooms which will help us to create the instance depending upon the user's input.

Again we don't have to change the client's code for adding the new facility.

Command design pattern

Command Pattern:

let us take up a design problem to understand command pattern. Suppose you are building a home automation system.

There is a programmable remote which can be used to turn on and off various items in your home like lights, stereo, AC etc.

It looks something like this.

You can do it with simple if-else statements like

```
if (buttonPressed == button1)
    lights.on()
```

But we need to keep in mind that turning on some devices like stereo comprises of many steps like setting cd, volume etc.

Also we can reassign a button to do something else.

By using simple if-else we are coding to implementation rather than interface.

Also there is tight coupling.

So what we want to achieve is a design that provides loose coupling and remote control should not have much information about a particular device.

The command pattern helps us do that.

Definition: The command pattern encapsulates a request as an object, thereby letting us parameterize other objects with different requests, queue or log requests, and support undoable operations.

The definition is a bit confusing at first but let's step through it.

In analogy to our problem above remote control is the client and stereo, lights etc. are the receivers.

In command pattern there is a Command object that *encapsulates a request* by binding together a set of actions on a specific receiver.

It does so by exposing just one method execute() that causes some actions to be invoked on the receiver.

Parameterizing other objects with different requests in our analogy means that the button used to turn on the lights can later be used to turn on stereo or maybe open the garage door.

queue or log requests, and support undoable operations means that Command's Execute operation can store state for reversing its effects in the Command itself.

The Command may have an added unExecute operation that reverses the effects of a previous call to execute.

It may also support logging changes so that they can be reapplied in case of a system crash.

Java implementation of remote control example:

```
// A simple Java program to demonstrate implementation of Command Pattern using a remote control example.
 // An interface for command
interface Command
{
   public void execute();
}

// Light class and its corresponding command
// classes
class Light
{
   public void on()
   {
      System.out.println("Light is on");
   }
   public void off()
   {
      System.out.println("Light is off");
   }
}
```

```
class LightOnCommand implements Command
{
    Light light;
     // The constructor is passed the light it is going to control.
    public LightOnCommand(Light light)
    {
       this.light = light;
    }
    public void execute()
    {
       light.on();
    }
}
class LightOffCommand implements Command
{
    Light light;
    public LightOffCommand(Light light)
    {
        this.light = light;
    }
    public void execute()
    {
         light.off();
    }
}
```

```
// Stereo and its command classes
class Stereo
{
   public void on()
   {
      System.out.println("Stereo is on");
   }
   public void off()
   {
      System.out.println("Stereo is off");
   }
   public void setCD()
   {
      System.out.println("Stereo is set " +  "for CD input");
   }
   public void setDVD()
   {
      System.out.println("Stereo is set"+   " for DVD input");
   }
   public void setRadio()
   {
      System.out.println("Stereo is set" +  " for Radio");
   }
   public void setVolume(int volume)
   {
     // code to set the volume
     System.out.println("Stereo volume set“ + " to " + volume);
   }
}
```

```
class StereoOffCommand implements Command
{
   Stereo stereo;
   public StereoOffCommand(Stereo stereo)
   {
      this.stereo = stereo;
   }
   public void execute()
   {
     stereo.off();
   }
}
class StereoOnWithCDCommand implements Command
{
    Stereo stereo;
    public StereoOnWithCDCommand(Stereo stereo)
    {
       this.stereo = stereo;
    }
    public void execute()
    {
       stereo.on();
       stereo.setCD();
       stereo.setVolume(11);
    }
}
```

```
// A Simple remote control with one button
class SimpleRemoteControl
{
   Command slot;  // only one button

   public SimpleRemoteControl()
   {
   }

   public void setCommand(Command command)
   {
      // set the command the remote will
      // execute
      slot = command;
   }

   public void buttonWasPressed()
   {
      slot.execute();
   }
}
```

```
// Driver class
class RemoteControlTest
{
   public static void main(String[] args)
   {
      SimpleRemoteControl remote =
              new SimpleRemoteControl();
      Light light = new Light();
      Stereo stereo = new Stereo();

      // we can change command dynamically
      remote.setCommand(new
               LightOnCommand(light));
      remote.buttonWasPressed();
      remote.setCommand(new
            StereoOnWithCDCommand(stereo));
      remote.buttonWasPressed();
      remote.setCommand(new
               StereoOffCommand(stereo));
      remote.buttonWasPressed();
    }
 }
```

Output:

```
Light is on
Stereo is on
Stereo is set for CD input
Stereo volume set to 11
Stereo is off
```

- Notice that the remote control doesn’t know anything about turning on the stereo.
- That information is contained in a separate command object. This reduces the coupling between them.

- Advantages:
- Makes our code extensible as we can add new commands without changing existing code.
- Reduces coupling between the invoker and receiver of a command.

- Disadvantages:
- Increase in the number of classes for each individual command

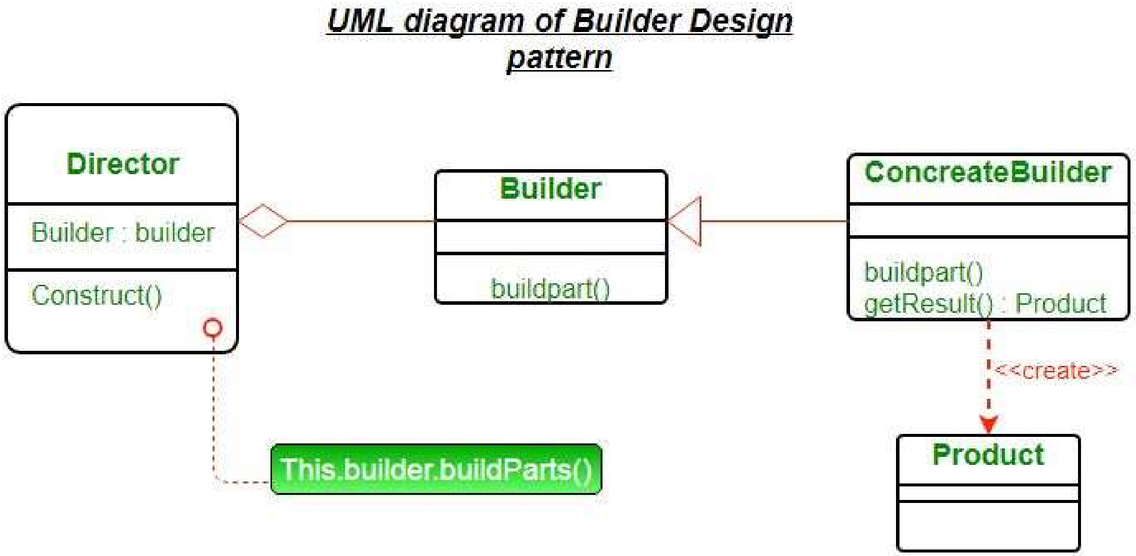
Builder design pattern

Builder pattern aims to "Separate the construction of a complex object from its representation so that the same construction process can create different representations."

It is used to construct a complex object step by step and the final step will return the object.

The process of constructing an object should be generic so that it can be used to create different representations of the same object.

UML Diagram of Builder Design Pattern


UML diagram of Builder Design pattern
Director
Builder : builder
Construct()
Builder
buildpart()
ConcreateBuilder
buildpart()
getResult() : Product
<<create>>
Product
This.builder.buildParts()

- Product – The product class defines the type of the complex object that is to be generated by the builder pattern.

- Builder – This abstract base class defines all of the steps that must be taken in order to correctly create a product.

- Each step is generally abstract as the actual functionality of the builder is carried out in the concrete subclasses. The GetProduct method is used to return the final product.

- The builder class is often replaced with a simple interface.

- **ConcreteBuilder –** There may be any number of concrete builder classes inheriting from Builder.
- These classes contain the functionality to create a particular complex product.
- **Director –** The director-class controls the algorithm that generates the final product object.
- A director object is instantiated and its Construct method is called.
- The method includes a parameter to capture the specific concrete builder object that is to be used to generate the product.
- The director then calls methods of the concrete builder in the correct order to generate the product object.
- On completion of the process, the GetProduct method of the builder object can be used to return the product.

Let’s see an Example of Builder Design Pattern :

Consider the construction of a home. Home is the final end product (object) that is to be returned as the output of the construction process.

It will have many steps like basement construction, wall construction, and so on roof construction.

Finally, the whole home object is returned.

Here using the same process you can build houses with different properties.

```
interface HousePlan
{
   public void setBasement(String basement);
   public void setStructure(String structure);
   public void setRoof(String roof);
   public void setInterior(String interior);
}
 class House implements HousePlan
{
   private String basement;
   private String structure;
   private String roof;
   private String interior;
     public void setBasement(String basement)
   {
       this.basement = basement;
   }
     public void setStructure(String structure)
   {
       this.structure = structure;
   }
     public void setRoof(String roof)
   {
       this.roof = roof;
   }
     public void setInterior(String interior)
   {
       this.interior = interior;
   }

}
```

```
interface HouseBuilder
{
    public void buildBasement();
    public void buildStructure();
     public void buildRoof();
    public void buildInterior();
    public House getHouse();
}
  class IglooHouseBuilder implements HouseBuilder
{
    private House house;
    public IglooHouseBuilder()
    {
        this.house = new House();
    }
      public void buildBasement()
    {
        house.setBasement("Ice Bars");
    }
      public void buildStructure()
    {
        house.setStructure("Ice Blocks");
    }
      public void buildInterior()
    {
        house.setInterior("Ice Carvings");
    }
      public void buildRoof()
    {
        house.setRoof("Ice Dome");
    }
      public House getHouse()
    {
        return this.house;
    }
}
```

```
class TipiHouseBuilder implements HouseBuilder
{
    private House house;
    public TipiHouseBuilder()
    {
        this.house = new House();
    }
      public void buildBasement()
    {
        house.setBasement("Wooden Poles");
    }
      public void buildStructure()
    {
        house.setStructure("Wood and Ice");
    }
      public void buildInterior()
    {
        house.setInterior("Fire Wood");
    }
      public void buildRoof()
    {
        house.setRoof("Wood, caribou and seal skins");
    }
      public House getHouse()
    {
        return this.house;
    }
```

```
class CivilEngineer
{

    private HouseBuilder houseBuilder;

    public CivilEngineer(HouseBuilder houseBuilder)
    {
        this.houseBuilder = houseBuilder;
    }

    public House getHouse()
    {
        return this.houseBuilder.getHouse();
    }

    public void constructHouse()
    {
        this.houseBuilder.buildBasement();
        this.houseBuilder.buildStructure();
        this.houseBuilder.buildRoof();
        this.houseBuilder.buildInterior();
    }
}
```

```
class Builder
{
    public static void main(String[] args)
    {
        HouseBuilder iglooBuilder = new IglooHouseBuilder();
        CivilEngineer engineer = new CivilEngineer(iglooBuilder);

        engineer.constructHouse();

        House house = engineer.getHouse();

        System.out.println("Builder constructed: "+ house);
    }
}
```

Output :

```
Builder constructed: House@6d06d69c
```

- Advantages of Builder Design Pattern
- The parameters to the constructor are reduced and are provided in highly readable method calls.
- Builder design pattern also helps in minimizing the number of parameters in the constructor and thus there is no need to pass in null for optional parameters to the constructor.
- Object is always instantiated in a complete state
- Immutable objects can be built without much complex logic in the object building process.

- Disadvantages of Builder Design Pattern

- The number of lines of code increases at least to double in builder pattern, but the effort pays off in terms of design flexibility and much more readable code.

- Requires creating a separate ConcreteBuilder for each different type of Product.

Abstract Factory Pattern

- Introduction

- Abstract Factory design pattern is one of the Creational pattern.

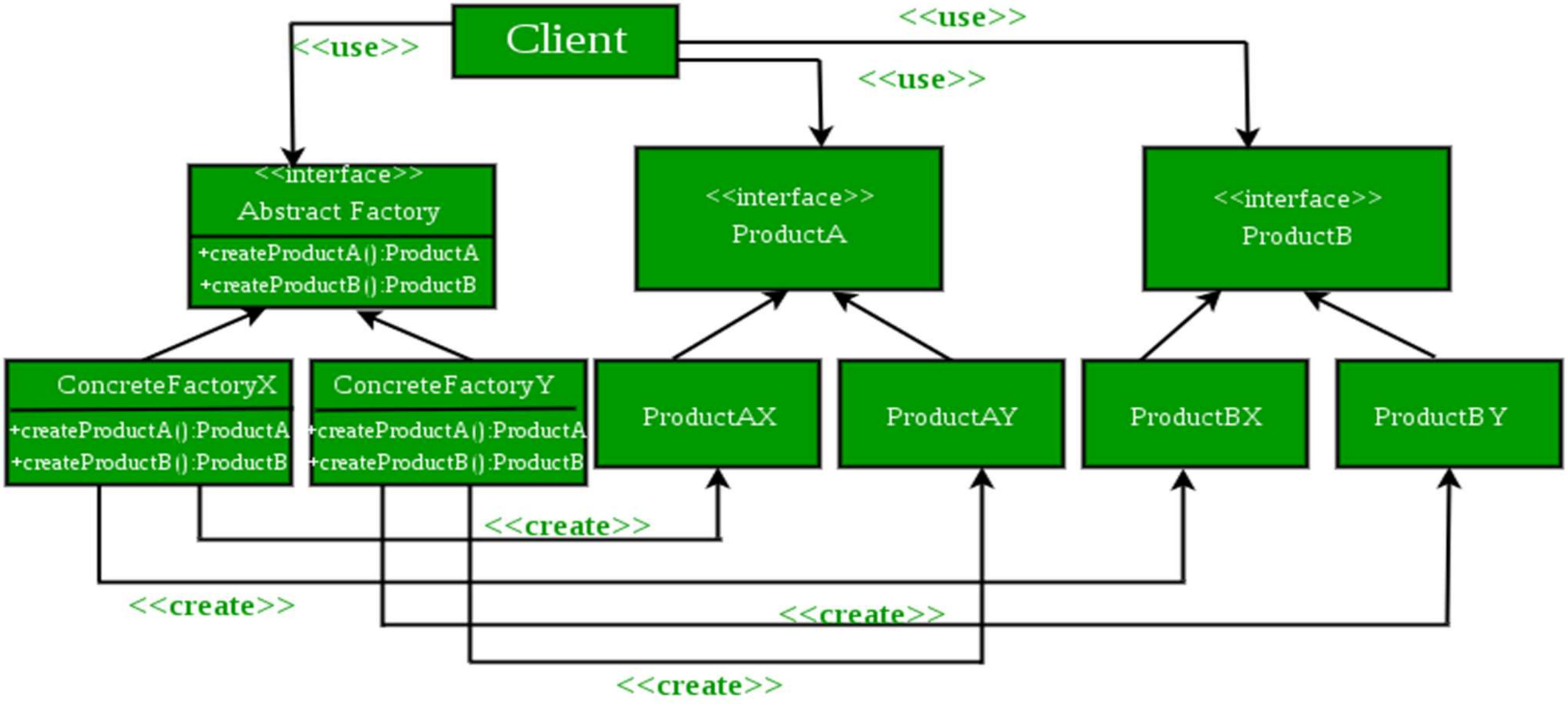
- Abstract Factory pattern is almost similar to Factory Pattern and is considered as another layer of abstraction over factory pattern.

- Abstract Factory patterns work around a super-factory which creates other factories.

- Abstract factory pattern implementation provides us with a framework that allows us to create objects that follow a general pattern.

- So at runtime, the abstract factory is coupled with any desired concrete factory which can create objects of the desired type.

- *UML class diagram example for the Abstract Factory Design Pattern.*


Client
<<use>>
<<use>>
<<use>>
<<interface>>
Abstract Factory
+createProductA():ProductA
+createProductB():ProductB
<<interface>>
ProductA
<<interface>>
ProductB
ConcreteFactoryX
+createProductA():ProductA
+createProductB():ProductB
ConcreteFactoryY
+createProductA():ProductA
+createProductB():ProductB
ProductAX
ProductAY
ProductBX
ProductBY
<<create>>
<<create>>
<<create>>
<<create>>

- AbstractFactory: Declares an interface for operations that create abstract product objects.

- ConcreteFactory: Implements the operations declared in the AbstractFactory to create concrete product objects.

- Product: Defines a product object to be created by the corresponding concrete factory and implements the AbstractProduct interface.

- Client: Uses only interfaces declared by AbstractFactory and AbstractProduct classes.

- Abstract Factory provides interfaces for creating families of related or dependent objects without specifying their concrete classes.

- Client software creates a concrete implementation of the abstract factory and then uses the generic interfaces to create the concrete objects that are part of the family of objects.

- The client does not know or care which concrete objects it gets from each of these concrete factories since it uses only the generic interfaces of their products.

- So with this idea of Abstract Factory pattern, we will now try to create a design that will facilitate the creation of related objects.

- Implementation

- Let’s take an example, Suppose we want to build a global car factory.

- If it was a factory design pattern, then it was suitable for a single location.

- But for this pattern, we need multiple locations and some critical design changes.

- We need car factories in each location like IndiaCarFactory, USACarFactory, and DefaultCarFactory.

- Now, our application should be smart enough to identify the location where it is being used, so we should be able to use the appropriate car factory without even knowing which car factory implementation will be used internally.

- This also saves us from someone calling the wrong factory for a particular location.

- Here we need another layer of abstraction that will identify the location and internally use the correct car factory implementation without even giving a single hint to the user.

- This is exactly the problem, which an abstract factory pattern is used to solve.

```
// Java Program to demonstrate the  working of Abstract Factory Pattern
 enum CarType
{
   MICRO, MINI, LUXURY
}
 abstract class Car
{
   Car(CarType model, Location location)
   {
      this.model = model;
      this.location = location;
   }
   abstract void construct();
   CarType model = null;
   Location location = null;
   CarType getModel()
   {
      return model;
   }
     void setModel(CarType model)
   {
      this.model = model;
   }
     Location getLocation()
   {
      return location;
   }
     void setLocation(Location location)
   {
      this.location = location;
   }
@Override
   public String toString()
   {
      return "CarModel - "+model + " located in "+location;
   }
}

class LuxuryCar extends Car
{
   LuxuryCar(Location location)
   {
      super(CarType.LUXURY, location);
      construct();
   }
   @Override
   protected void construct()
   {
      System.out.println("Connecting to luxury car");
   }
}
```

```
class MicroCar extends Car
{
    MicroCar(Location location)
    {
        super(CarType.MICRO, location);
        construct();
    }
    @Override
    protected void construct()
    {
        System.out.println("Connecting to Micro Car ");
    }
}
```

```
class MiniCar extends Car
{
   MiniCar(Location location)
   {
      super(CarType.MINI,location );
      construct();
   }

   @Override
   void construct()
   {
      System.out.println("Connecting to Mini car");
   }
}
```

```
enum Location
{
 DEFAULT, USA, INDIA
}
 class INDIACarFactory
{
   static Car buildCar(CarType model)
   {
      Car car = null;
      switch (model)
      {
         case MICRO:
            car = new MicroCar(Location.INDIA);
            break;

         case MINI:
            car = new MiniCar(Location.INDIA);
            break;

         case LUXURY:
            car = new LuxuryCar(Location.INDIA);
            break;

            default:
            break;
     }
      return car;
   }
}
```

```
class DefaultCarFactory
{
    public static Car buildCar(CarType model)
    {
        Car car = null;
        switch (model)
        {
            case MICRO:
                car = new MicroCar(Location.DEFAULT);
                break;

            case MINI:
                car = new MiniCar(Location.DEFAULT);
                break;

            case LUXURY:
                car = new LuxuryCar(Location.DEFAULT);
                break;

                default:
                break;

        }
        return car;
    }
}
```

```
class USACarFactory
{
   public static Car buildCar(CarType model)
   {
      Car car = null;
      switch (model)
      {
         case MICRO:
            car = new MicroCar(Location.USA);
            break;

         case MINI:
            car = new MiniCar(Location.USA);
            break;

         case LUXURY:
            car = new LuxuryCar(Location.USA);
            break;

            default:
            break;

      }
      return car;
   }
}
```

```
class CarFactory
{
   private CarFactory()
   {

   }
   public static Car buildCar(CarType type)
   {
      Car car = null;
      // We can add any GPS Function here which  read location property somewhere from configuration and use location specific car factory  Currently I'm just
     / /using INDIA as Location
      Location location = Location.INDIA;
            switch(location)
      {
         case USA:
            car = USACarFactory.buildCar(type);
            break;

         case INDIA:
            car = INDIACarFactory.buildCar(type);
            break;

         default:
            car = DefaultCarFactory.buildCar(type);

      }

      return car;

   }
}
```

```
class AbstractDesign
{
    public static void main(String[] args)
    {
        System.out.println(CarFactory.buildCar(CarType.MICRO));
        System.out.println(CarFactory.buildCar(CarType.MINI));
        System.out.println(CarFactory.buildCar(CarType.LUXURY));
    }
}
```

Output :

```
Connecting to Micro Car
CarModel - MICRO located in INDIA
Connecting to Mini car
CarModel - MINI located in INDIA
Connecting to luxury car
CarModel - LUXURY located in INDIA
```

- Difference

- The main difference between a “factory method” and an “abstract factory” is that the factory method is a single method, and an abstract factory is an object.

- The factory method is just a method, it can be overridden in a subclass, whereas the abstract factory is an object that has multiple factory methods on it.

- The Factory Method pattern uses inheritance and relies on a subclass to handle the desired object instantiation.

- Advantages: This pattern is particularly useful when the client doesn’t know exactly what type to create.

- Isolation of concrete classes: The Abstract Factory pattern helps you control the classes of objects that an application creates.

- Because a factory encapsulates the responsibility and the process of creating product objects, it isolates clients from implementation classes.

- Clients manipulate instances through their abstract interfaces.

- Product class names are isolated in the implementation of the concrete factory; they do not appear in client code.

- Exchanging Product Families easily: The class of a concrete factory appears only once in an application, that is where it's instantiated.
- This makes it easy to change the concrete factory an application uses.
- It can use various product configurations simply by changing the concrete factory.
- Because an abstract factory creates a complete family of products, the whole product family changes at once.
- Promoting consistency among products: When product objects in a family are designed to work together, it's important that an application use objects from only one family at a time.
- AbstractFactory makes this easy to enforce.

- Disadvantages

- Difficult to support new kinds of products: Extending abstract factories to produce new kinds of Products isn’t easy.

- That’s because the AbstractFactory interface fixes the set of products that can be created.

- Supporting new kinds of products requires extending the factory interface, which involves changing the AbstractFactory class and all of its subclasses.

- When to use

- When the system must be independent in terms of how its objects are created, composed, and represented.

- This constraint must be enforced when a family of related objects needs to be used together.

- When you want to provide an object library that does not show implementations but only exposes interfaces.

- When the system has to be configured with one of several families of objects.

Prototype Design Pattern

- Prototype allows us to hide the complexity of making new instances from the client.
- The concept is to copy an existing object rather than creating a new instance from scratch, something that may include costly operations.
- The existing object acts as a prototype and contains the state of the object.
- The newly copied object may change same properties only if required.
- This approach saves costly resources and time, especially when object creation is a heavy process.
- The prototype pattern is a creational design pattern.
- Prototype patterns are required, when object creation is time consuming, and costly operation, so we create objects with the existing object itself.

- One of the best available ways to create an object from existing objects is the clone() method.
- Clone is the simplest approach to implement a prototype pattern.
- However, it is your call to decide how to copy existing object based on your business model.
- Prototype Design Participants
- 1) Prototype : This is the prototype of an actual object.
- 2) Prototype registry : This is used as a registry service to have all prototypes accessible using simple string parameters.
- 3) Client : Client will be responsible for using registry service to access prototype instances.

- When to use the Prototype Design Pattern

- When a system should be independent of how its products are created, composed, and represented and When the classes to instantiate are specified at run-time.

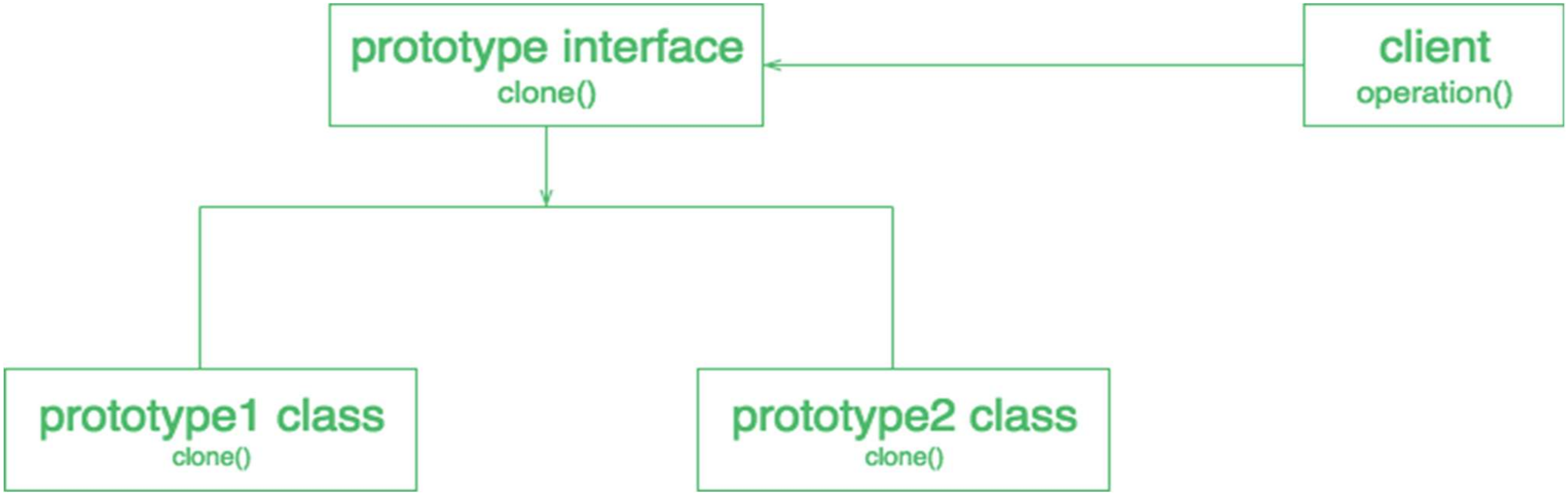
- For example,

- 1) By dynamic loading or To avoid building a class hierarchy of factories that parallels the class hierarchy of products or

- 2) When instances of a class can have one of only a few different combinations of state.

- It may be more convenient to install a corresponding number of prototypes and clone them rather than instantiating the class manually, each time with the appropriate state.

- The UML Diagram of the Prototype Design Pattern


prototype interface
clone()
client
operation()
prototype1 class
clone()
prototype2 class
clone()

```java
// A Java program to demonstrate working of  Prototype Design Pattern with example of a ColorStore class to store existing objects.
import java.util.HashMap;
import java.util.Map;
// Cloneable interface allows the implementing class to  have its objects to be cloned instead of using a new operator.
// It is available in Java.lang.Cloneable.

abstract class Color implements Cloneable
{
    protected String colorName;

    abstract void addColor();

    public Object clone()
    {
        Object clone = null;
        try
        {
            clone = super.clone();
        }
        catch (CloneNotSupportedException e)
        {
            e.printStackTrace();
        }
        return clone;
    }
}
```

```
class blueColor extends Color
{
    public blueColor()
    {
        this.colorName = "blue";
    }

    @Override
    void addColor()
    {
        System.out.println("Blue color added");
    }

}

class blackColor extends Color{

    public blackColor()
    {
        this.colorName = "black";
    }

    @Override
    void addColor()
    {
        System.out.println("Black color added");
    }
}
```

```
class ColorStore {

    private static Map<String, Color> colorMap = new HashMap<String, Color>();

    static
    {
        colorMap.put("blue", new blueColor());
        colorMap.put("black", new blackColor());
    }

    public static Color getColor(String colorName)
    {
        return (Color) colorMap.get(colorName).clone();
    }
}

// Driver class
class Prototype
{
    public static void main (String[] args)
    {
        ColorStore.getColor("blue").addColor();
        ColorStore.getColor("black").addColor();
        ColorStore.getColor("black").addColor();
        ColorStore.getColor("blue").addColor();
    }
}
```

Output:

```
Blue color added
Black color added
Black color added
Blue color added
```

- Advantages of Prototype Design Pattern
- **Adding and removing products at run-time –** Prototypes let you incorporate a new concrete product class into a system simply by registering a prototypical instance with the client.
- That's a bit more flexible than other creational patterns, because a client can install and remove prototypes at run-time.
- **Specifying new objects by varying values –** Highly dynamic systems let you define new behavior through object composition by specifying values for an object's variables and not by defining new classes.
- **Specifying new objects by varying structure –** Many applications build objects from parts and subparts.
- For convenience, such applications often let you instantiate complex, user-defined structures to use a specific subcircuit again and again.
- **Reduced subclassing –** Factory Method often produces a hierarchy of Creator classes that parallels the product class hierarchy.
- The Prototype pattern lets you clone a prototype instead of asking a factory method to make a new object.
- Hence you don't need a Creator class hierarchy at all.

- Disadvantages of Prototype Design Pattern

- Overkill for a project that uses very few objects and/or does not have an underlying emphasis on the extension of prototype chains.

- It also hides concrete product classes from the client

- Each subclass of Prototype must implement the clone() operation which may be difficult, when the classes under consideration already exist.

- Also implementing clone() can be difficult when their internals include objects that don’t support copying or have circular references.

Proxy Design Pattern

- Proxy means ‘in place of’, representing’ or ‘in place of’ or ‘on behalf of’ are literal meanings of proxy and that directly explains Proxy Design Pattern.

- Proxies are also called surrogates, handles, and wrappers.

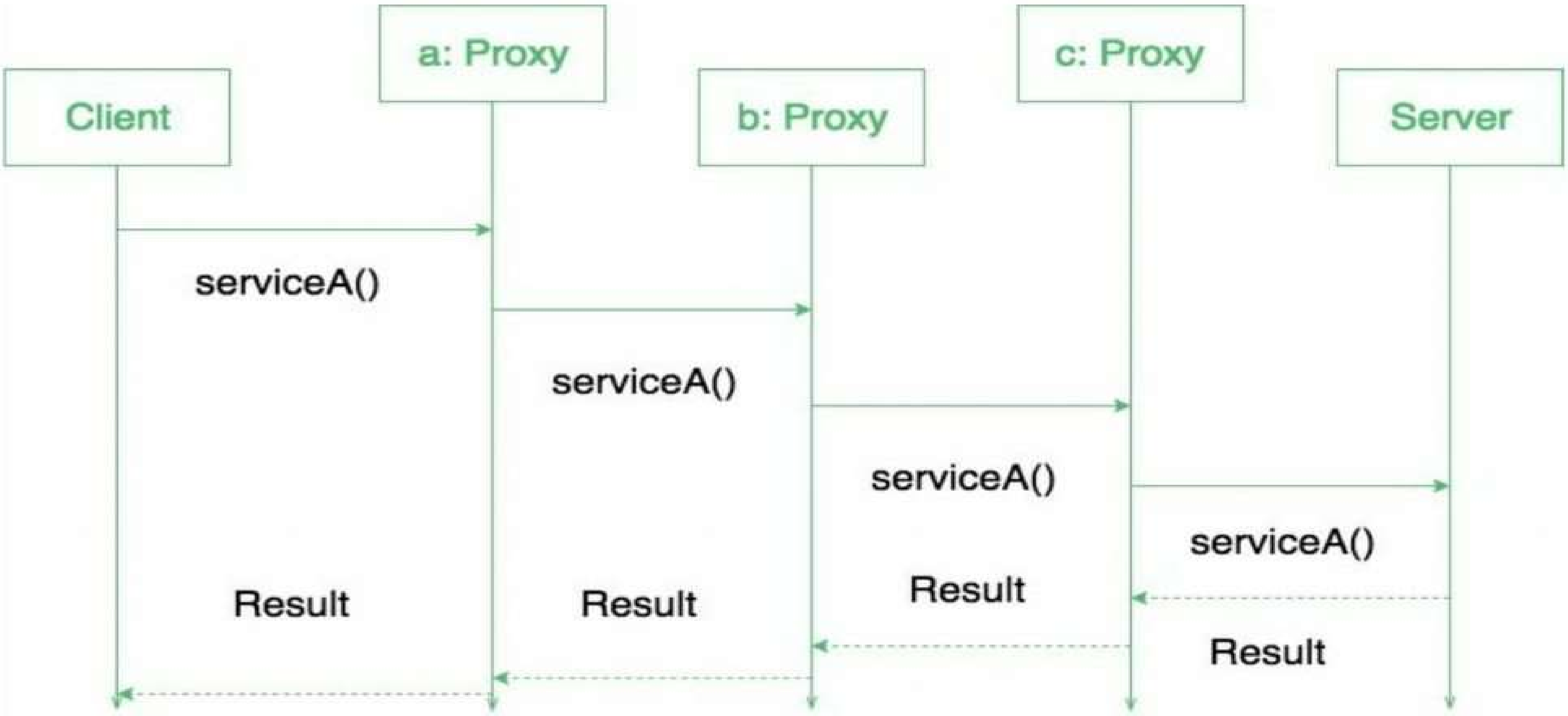
- A real world example can be a cheque or credit card is a proxy for what is in our bank account.

- It can be used in place of cash, and provides a means of accessing that cash when required.

- And that’s exactly what the Proxy pattern does – “Controls and manage access to the object they are protecting“.

- Behavior
- As in the decorator pattern, proxies can be chained together.

- The client, and each proxy, believes it is delegating messages to the real server:


a: Proxy
c: Proxy
Client
b: Proxy
Server
serviceA()
serviceA()
serviceA()
serviceA()
Result
Result
Result
Result

- When to use this pattern?
- Proxy pattern is used when we need to create a wrapper to cover the main object's complexity from the client.
- Types of proxies
- Remote proxy:
- They are responsible for representing the object located remotely.
- Talking to the real object might involve marshalling and unmarshalling of data and talking to the remote object.
- All that logic is encapsulated in these proxies and the client application need not worry about them.

- Virtual proxy:
- These proxies will provide some default and instant results if the real object is supposed to take some time to produce results.
- These proxies initiate the operation on real objects and provide a default result to the application.
- Once the real object is done, these proxies push the actual data to the client where it has provided dummy data earlier.
- Protection proxy:
- If an application does not have access to some resource then such proxies will talk to the objects in applications that have access to that resource and then get the result back.

- Smart Proxy:

- A smart proxy provides additional layer of security by interposing specific actions when the object is accessed.

- An example can be to check if the real object is locked before it is accessed to ensure that no other object can change it.

- Some Examples

- A very simple real life scenario is our college internet, which restricts few site access.

- The proxy first checks the host you are connecting to, if it is not part of restricted site list, then it connects to the real internet.

- This example is based on Protection proxies.

- Interface of Internet
- package com.bits.demo.proxy;
-   public interface Internet
- {
-     public void connectTo(String serverhost) throws Exception;
- }

- RealInternet.java

```
package com.bits.demo.proxy;

public class RealInternet implements Internet
{
    @Override
    public void connectTo(String serverhost)
    {
        System.out.println("Connecting to "+ serverhost);
    }
}
```

- ProxyInternet.java

```java
package com.bits.demo.proxy;
import java.util.ArrayList;
import java.util.List;
public class ProxyInternet implements Internet
{
        private Internet internet = new RealInternet();
        private static List<String> bannedSites;
        static
        {
                bannedSites = new ArrayList<String>();
                bannedSites.add("abc.com");
                bannedSites.add("def.com");
                bannedSites.add("ijk.com");
                bannedSites.add("lnm.com");
        }
```

- ProxyInternet.java

```
• @Override
•       public void connectTo(String serverhost) throws Exception
•       {
•               if(bannedSites.contains(serverhost.toLowerCase()))
•               {
•                       throw new Exception("Access Denied");
•               }
•
•               internet.connectTo(serverhost);
•       }

• }
```

- Client.java

```
package com.bits.demo.proxy;
public class Client
{
        public static void main (String[] args)
        {
                Internet internet = new ProxyInternet();
                try
                {
                        internet.connectTo(“bits-pilani.ac.in");
                        internet.connectTo("abc.com");
                }
                catch (Exception e)
                {
                        System.out.println(e.getMessage());
                }
        }
}
```

- As one of the site is mentioned in the banned sites, So

- Running the program will give the output :

- Connecting to bits-pilani.ac.in
- Access Denied

- Benefits:

- One of the advantages of Proxy pattern is security.

- This pattern avoids duplication of objects which might be huge size and memory intensive.

- This in turn increases the performance of the application.

- The remote proxy also ensures about security by installing the local code proxy (stub) in the client machine and then accessing the server with help of the remote code.

- Drawbacks/Consequences:
- This pattern introduces another layer of abstraction which sometimes may be an issue if the RealSubject code is accessed by some of the clients directly and some of them might access the Proxy classes.
- This might cause disparate behaviour.
- Interesting points:
- There are few differences between the related patterns.
- Like Adapter pattern gives a different interface to its subject, while Proxy patterns provides the same interface from the original object but the decorator provides an enhanced interface.
- Decorator pattern adds additional behaviour at runtime.
- Proxy used in Java API: java.rmi.*;

BITS Pilani
Pilani|Dubai|Goa|Hyderabad

BSDCHZC355
Statistical Inference & Applications

Shaibal Kr. Sen
Session 01

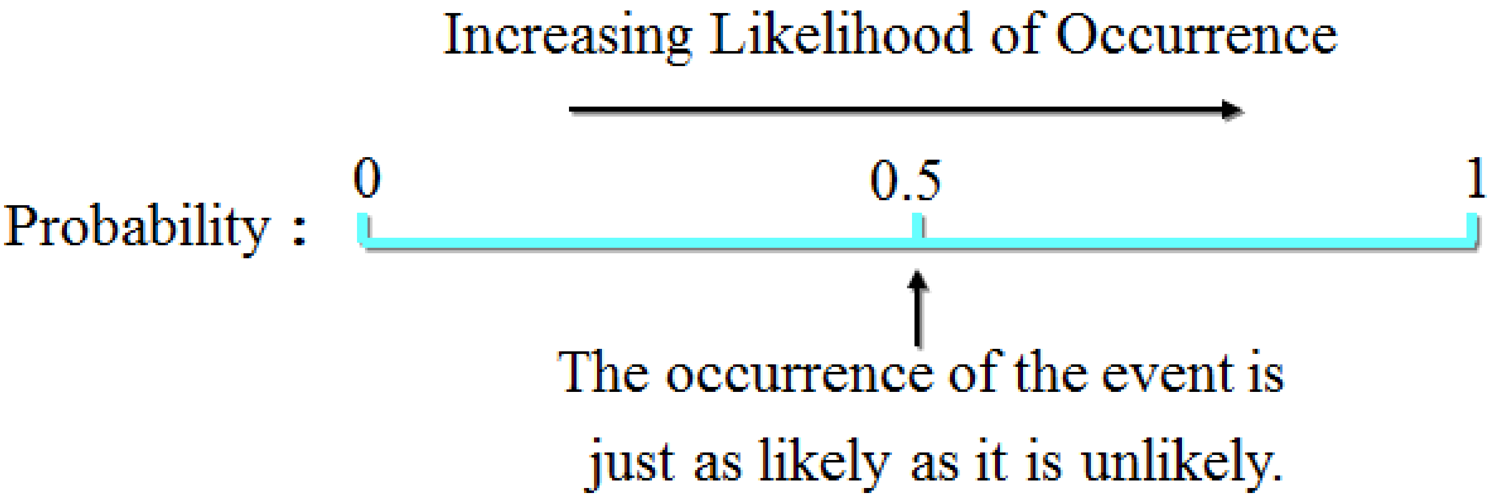
STATISTICAL INFERENCES
&
APPLICATIONS

Scheduled Topics to be covered

Review of **P**robability & **S**tatistics

Probability **D**istribution - Z, t , F distributions

Probability as a Numerical Measure of the Likelihood of Occurrence (Uncertainty).


Increasing Likelihood of Occurrence
Probability :
0
0.5
1

The occurrence of the event is just as likely as it is unlikely.

Classical Probability : Probability is a numerical measure of the likelihood that an event will occur.

Probability of an Event = No. of Favourable outcomes / Total No. of Possible outcomes

Empirical Probability : Empirical or Relative Frequency is the 2nd type of objective probability. It is based on the number of times an event occurs as a proportion of a known no. of trials.

Mathematically,

$$\text{Empirical Probability} = \frac{\text{No. of times the Event occurs}}{\text{Total no. of observations}}$$

The empirical approach to probability is based on what is called the law of large numbers. The key to establishing probabilities empirically is that more observations will provide a more accurate estimate of the probability.

Law of large nos. : Over a large no. of trials, the empirical probability of an event will approach its true probability.

Axioms of Probability

Probability is a number that is assigned to each member of a collection of events from a random experiment that satisfies the following properties :

If S is the sample space and E is any event in a random experiment,

(1) $P(S) = 1$

(2) $0 \leq P(E) \leq 1$

(3) For two mutually exclusive events events E_1 and E_2; $E_1 \cap E_2 = \phi$ (Null set)

and $P(E_1 U E_2) = P(E_1) + P(E_2) – P(E_1 \cap E_2) = P(E_1) + P(E_2)$

Ex 1 **:** In a certain residential hub, 60% of all households get internet service from the local cable company,80% get the television service from that company, and 50% get both services from that company.

If a household is randomly selected, what is the probability that it gets at least one of these two services from the company, and what is the probability that it gets exactly one of these services from the company?

Soln **:** Let household getting Internet services be A, getting TV services be B and getting both TV & Internet services will be $A \cap B$. Then as per given data –

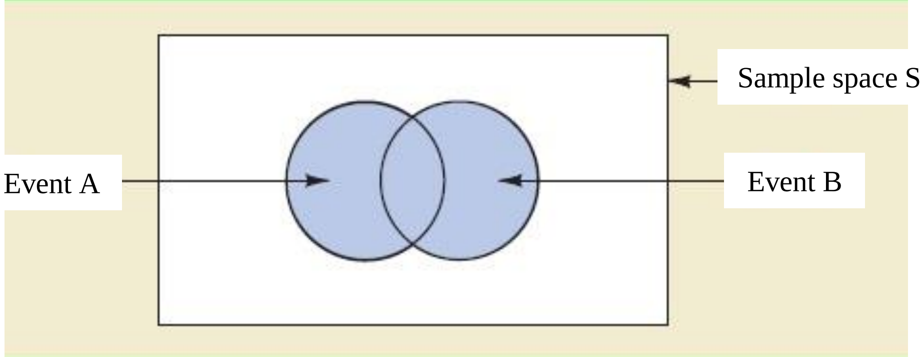
$P(A) = 0.6$, $P(B) = 0.8$ and $P(A \cap B) = 0.5$.

Required to find P(exactly one of the services)

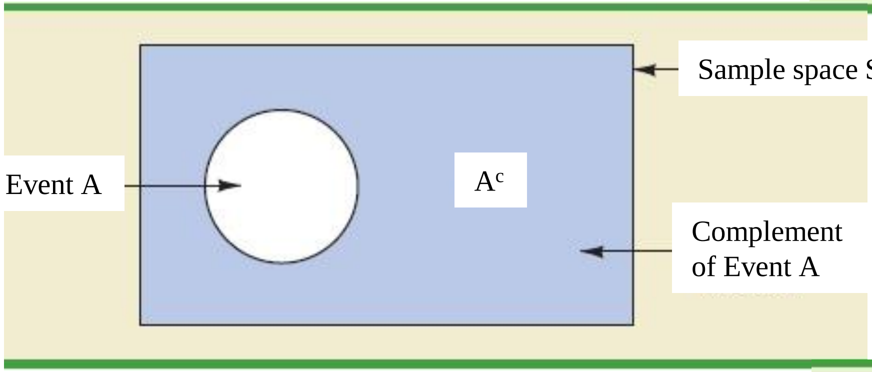
We know, $P(AUB) = P(A) + P(B) – P(A \cap B) = 0.6 + 0.8 – 0.5 = 0.9$

so, P(exactly one of the services) = $P(AUB) - P(A \cap B) = 0.9 – 0.5 = 0.4$

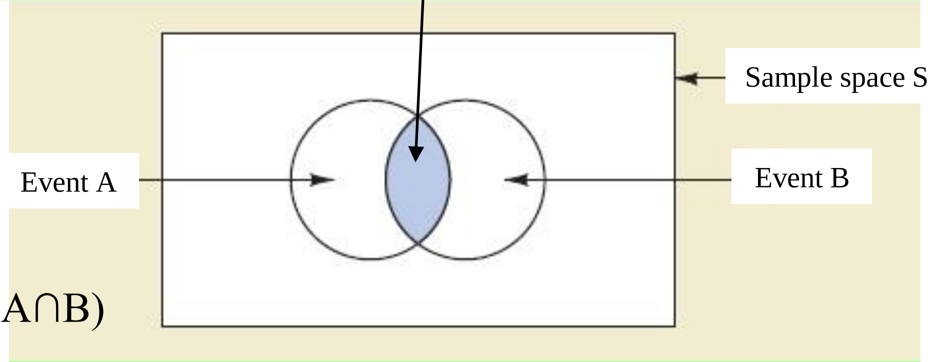
Refer the Venn diagram in next slide


Sample space S
Event A
Event B

P(A) = 1 – P(A^c). It leads to P(A^c) = 1 – P(A)


Sample space S
Event A
A^c
Complement of Event A

P(A∩B)


Sample space S
Event A
Event B

P(AUB) = P(A) + P(B) – P(A∩B)

P(exactly one of the services) = P(AUB) - P(A∩B)

Conditional Probability

Definition : For any two events A and B with $P(B) > 0$, the conditional probability of A given that B has occurred is defined by –

$$P(A|B) = P(A\cap B) / P(B)$$

The definition of conditional probability yields the following result, obtained by multiplying both sides of equation by P(B).

Multiplication Rule : $P(A\cap B) = P(A|B).P(B)$

This rule is important because it is often the case that $P(A\cap B)$ is desired, whereas both P(B) and $P(A|B)$ can be specified from the problem description.

Independent Events

If A and B are independent events then

$P(A|B) = P(A \cap B) / P(B) = P(A).P(B) / P(B) = P(A)$

and

$P(B|A) = P(A \cap B) / P(A) = P(A).P(B) / P(A) = P(B)$

Ex 2 **:** Suppose a card is drawn from a pack of 52 cards. The probability of it being a card of diamond (D) is 13/52 = 1/ 4. Now, suppose that it is told that the card drawn is a red card (R), would the probability of it being a D card be 1/ 4 as before or will it be different given the additional information that the card drawn is R.

Soln **:** $P(D/R) = P(D \cap R) / P(R) = (13/52)/(26/52) = (1/4)/(1/2) = 1/2$

Here, Diamond (D) is a subset of Red (R) ie. $D \sqsubset R$

$n(D) = 13$, $n(R) = 26$, $n(D \cap R) = 13$

Bayes’ Theorem

Let A_1, A_2, ….., A_k be a collection of k mutually exclusive and exhaustive events with prior probabilities $P(A_i)$ where i = 1, 2, …., k. Then for any other event B for which $P(B) > 0$, the posterior probability of A_j given that B has occurred is –

$$P(A_j|B) = P(A_j \cap B) / P(B) = P(B|A_j).P(A_j) / \sum_{i=1}^{k} P(B|A_i).P(A_i)$$

$$j = 1, 2, …., k$$

Ex 3 **:** Three persons A,B and C are competing for the post of CEO of a company. The chances of they becoming CEO are 0.2, 0.3 and 0.5 respectively. The chances of they taking employees beneficial decisions are 0.50, 0.45 and 0.6 respectively.

What are the chances of having employees beneficial decisions after having new CEO.

If employees beneficial decision has been taken, what is the probability that it has been taken by A as new CEO. Similarly, B and C as new CEOs.

Soln :

$$P(\text{Ben}) = 0.2 \times 0.5 + 0.3 \times 0.45 + 0.5 \times 0.6$$

$$= 0.10 + 0.135 + 0.30 = 0.535$$

P(A|Ben) = P(A).P(Ben|A) / P(Ben) = 0.2 x 0.5 / 0.535 = 0.1869

P(B|Ben) = P(B).P(Ben|B) / P(Ben) = 0.3 x 0.45 / 0.535 = 0.2523

P(C|Ben) = P(C).P(Ben|C) / P(Ben) = 0.5 x 0.6 / 0.535 = 0.5607

Statistics

Statistics may be defined as science that is employed to –

- Collect the data
- Organize and Present the data in a systematic manner
- Analyse the data
- Infer about the data
- Take decision from the data.

Statistics may be defined as numerical data with a view to analyse it.

Main Branches of Statistics

Descriptive Statistics **:** Used to study the patterns of past and present data. Classification, tabulation, visualization, central tendency and dispersion measures etc. are some of the important tools of Descriptive Statistics.

Inferential Statistics **:** Used to estimate and predict the behavior of Big Data (Population) with the help of Small Data (Sample). Sampling, Estimation, Testing of Hypotheses are some of the important tools of Inferential Statistics.

Mean (Important Measure of Central Tendency)

Also referred as the “**Arithmetic Average**”.

The most commonly used measure of the Center of Data.

Numbers that describe what is average or typical of the distribution.

Computation of Sample Mean :

$\overline{Y} = \Sigma Y / N = (Y_1 + Y_2 + Y_3 + \ldots + Y_n) / N$ where “Y bar” equals the sum of all the scores, Y, divided by the number of scores, N.

Computation of the Mean for grouped Data

$\overline{Y} = \Sigma f_i y_i / N$, where $f_i y_i$ = a score multiplied by its frequency, $N = \Sigma f_i$.

The Standard Deviation (Important measure of Dispersion)

Most common and most important measure of variability is the standard deviation

- A measure of the standard, or average, distance from the mean
- Describes whether the scores are clustered closely around the mean or are widely scattered.

Calculation differs for population and samples.

Variance is a necessary *companion concept* to standard deviation but *not the same* concept.

For Population Variance is $\sigma^2 = \sqrt{}\{(x_i - \overline{x})^2\} / N$, Standard Deviation $\sigma = \sqrt{}(\text{Variance}) = \sqrt{}[\{\Sigma(x_i - \overline{x})^2\} / N]$.

For Sample Variance is $S^2 = \{\Sigma(x_i - \overline{x})^2\}/ (n - 1)$ and Standard Deviation $S = \sqrt{}[\{\Sigma(x_i - \overline{x})^2\}/ (n - 1)]$.

Ex 4 : Computation of Deviations & Squared Deviations about Mean for the class size Data

	No. of students in a Class (x_i)	Deviation about Mean ($x_i - \overline{x}$)	Squared deviation about Mean $(x_i - \overline{x})^2$
	46	2	4
	54	10	100
	42	-2	4
	46	2	4
	32	-12	144
Total	220	0	256

Mean $\overline{x} = 220 / 5 = 44$

$$S^2 = \Sigma(x_i - \overline{x})^2 / (n - 1) = 256 / (5 - 1) = 64$$

$$S = \sqrt{[\Sigma(x_i - \overline{x})^2 / (n - 1)]} = \sqrt{64} = 8 \text{ (Positive square root)}$$

Random Variable

Random Variable (RV) **:** A random variable is a real valued function which is a mapping from the sample space Ω to the set of real numbers, ie., $X : \Omega \rightarrow R$. There are two types of RV viz. – (i) Discrete RV and (ii) Continuous RV

(i) Discrete RV take on countable numbers (may be finite or countable infinite values) i.e., without decimal like Natural numbers, Whole numbers, Integers etc.

(ii) Continuous RV take on any values in an interval i.e., in the set of real numbers which includes, negative, positive, rational, irrational, decimal etc.

Probability Distributions based on RVs

Two types of Probability distributions are –

(1) Discrete Probability Distribution **:** A probability distribution based on discrete RV is called discrete probability distribution. p(X) satisfies -

(i) $0 \leq p(x) \leq 1$, for all x

(ii) $\sum_{\text{All } x_i} p(x) = 1$

(2) Continuous Probability Distribution : A probability distribution based on continuous RV is called continuous probability distribution. An f(x) is called a probability density function of a continuous random variable if it satisfy the following properties –

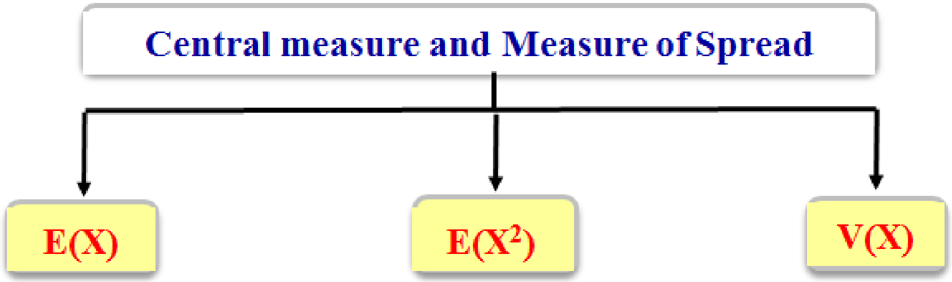
(i) $f(x) \geq 0,\ x \in (a, b)$

(ii) $\int_a^b f(x)\,dx = 1$

(iii) $P(a \leq X \leq b) = \int_a^b f(x)\,dx$

Expected value and Variance (Probability Distribution)

Like mean and standard deviation are computed to describe data measured by quantitative variable, a similar measures viz., expected value (mean) and variance for random variable X are computed for describing the probability distribution using the formula


Central measure and Measure of Spread

$E(X)$ $E(X^2)$ $V(X)$

For a discrete random variable X with probability mass function p(x),

Mean or Expected value $= \mu = E(X) = \sum_{\text{over all } x} xp(x)$

$E(X^2) = \sum_{\text{over all } x} x^2 p(x)$

Variance $= V(x) = \sigma^2 = E[(x-\mu)^2] = \sum (x-\mu)^2 p(x) = E(x^2) - (E(x))^2$

Standard deviation $\sigma = \sqrt{V(x)}$

Ex 5 **:** For the given values of p(x) for various values of x find the Expected value, Variance and p(x) for $x < 2$ and $x > 4$.

$$p(x) = \begin{cases} 0, & x < 0 \\ 0.06, & x = 0 \\ 0.13, & x = 1 \\ 0.20, & x = 2 \\ 0.28, & x = 3 \\ 0.25, & x = 4 \\ 0.05, & x = 5 \\ 0.03, & x = 6 \end{cases}$$

Soln **:** Mean $\mu = E(X) = 0.06 \times 0 + 0.13 \times 1 + 0.2 \times 2 + 0.28 \times 3 + 0.25 \times 4 + 0.05 \times 5 + 0.03 \times 6$

$= 0.13 + 0.4 + 0.84 + 1 + 0.25 + 0.18 = 2.8$

$E(X^2) = 0.06 \times 0^2 + 0.13 \times 1^2 + 0.2 \times 2^2 + 0.28 \times 3^2 + 0.25 \times 4^2 + 0.05 \times 5^2 + 0.03 \times 6^2$

$= 0.13 + 0.8 + 2.52 + 4 + 1.25 + 1.08 = 9.78$

$V(X) = E(X^2) – (E(X))^2 = 9.78 – 2.8^2 = 1.94$

$SD(X) = \sqrt{1.94} = 1.392$

Now, $P(X < 2) = P(x = 1) + P(x = 0) = 0.13 + 0.06 = 0.19$

$P(X > 4) = P(x = 6) + P(x = 5) = 0.03 + 0.05 = 0.08$

As in case of discrete probability distribution, the expected value E(X) and variance V(X) can be computed for the continuous probability distribution –

$E(X) = \int_{-\infty}^{\infty} x f(x)\, dx$ $E(X^2) = \int_{-\infty}^{\infty} x^2 f(x)\, dx$

$V(X) = E(X^2) - (E(X))^2$ $SD = \sqrt{V(X)}$

Ex 6 : Let x be a random variable with pdf given by –

$$f(x) = \begin{cases} cx^2 & \text{for } -1 \le x \le 1 \\ 0, & \text{otherwise} \end{cases}$$

(i) Find constant ‘c’

(ii) Find E(x)

(iii) Find V(x)

(iv) Find $P(x \ge 1/2)$

Soln : (i) We know Total Probability = $\int_{-\infty}^{\infty} f(x)\, dx = 1$ and as per given data

$c\int_{-1}^{1} cx^2\, dx = 1.$ Hence, $c\,(x^3 / 3)\Big|_{-1}^{1} = 1$ or, $c\,\{1 / 3 - (-1)^3 / 3\} = 1$

or, $c\,(1 / 3 + 1 / 3) = 1$ ie. $c = 3 / 2 = 1.5$

Therefore, $f(x) = cx^2 = 1.5x^2$ for $-1 \le x \le 1$

(ii) $\mu = E(X) = \int_{-1}^{1} xf(x)\, dx = \int_{-1}^{1} x(1.5x^2)\, dx = 1.5\int_{-1}^{1} x^3\, dx = 1.5[x^4 / 4]_{-1}^{1}$

$= 1.5(1 / 4 - 1 / 4) = 0$

(iii) $V(X) = E(X^2) - (E(X))^2$

$E(X^2) = \int_{-1}^{1} x^2 f(x)\, dx = \int_{-1}^{1} x^2(1.5x^2)\, dx = 1.5(x^5 / 5)\Big|_{-1}^{1} = 1.5(1 / 5 + 1 /5) = 0.6$

So, $V(X) = 0.6 - 0^2 = 0.6$

(iv) $P(x \ge 1 / 2) = \int_{1/2}^{1} f(x)\, dx = \int_{1/2}^{1} (1.5x^2)\, dx = 1.5[x^3 / 3]_{1/2}^{1} = 1.5(0.875) / 3 = 0.4375$

Binomial Distribution

A random variable ‘X’ is said to have Binomial distribution if its probability mass function is given by

$$P(x) = \begin{cases} {}^{n}C_{x}\, p^{x}\, q^{n-x}, & x = 0, 1, 2, 3, \ldots\ldots, n \\ 0, \text{ elsewhere} \end{cases}$$

Binomial distribution is a discrete probability distribution.

Binomial distribution will be applied under the following experimental conditions.

1) The number of trials (n) is finite.

2) The trails are independent of each other.

3) The probability of success p is constant for each trail.

4) Each trail results in two mutually exclusive events known as success and failure.

Ex 6 **:** The probability that a man aged 60 years will remain alive till 70 is 0.65. Find – (i) the probability that out of 10 such men at least 7 would be alive at 70 and (ii) Exactly 4 out of 10 would be alive till 70

Soln **:** We know, $P(x) = {}^{n}C_{x}\, p^{x}\, q^{n-x}$, $x = 0, 1, 2, 3, \ldots\ldots, n$

where, $p = 0.65$; so, $q = 1 - 0.65 = 0.35$; $n = 10$

(i) Required to find $P(x \geq 7) = P(x = 7) + P(x = 8) + P(x = 9) + P(x = 10)$

$= {}^{10}C_{7}\,(0.65)^{7}\,(0.35)^{10-7} + {}^{10}C_{8}\,(0.65)^{8}\,(0.35)^{10-8} + {}^{10}C_{9}\,(0.65)^{9}\,(0.35)^{10-9} +$
${}^{10}C_{10}\,(0.65)^{10}\,(0.35)^{10-10}$

$= {}^{10}C_{7}\,(0.65)^{7}\,(0.35)^{3} + {}^{10}C_{8}\,(0.65)^{8}\,(0.35)^{2} + {}^{10}C_{9}\,(0.65)^{9}\,(0.35)^{1} + {}^{10}C_{10}$
$(0.65)^{10}\,(0.35)^{0}$

$= 10! / (7! \text{ X } 3!)\{(0.65)^7(0.35)^3\} + 10! / (8! \text{ X } 2!)\{(0.65)^8(0.35)^2\} + 10! / (9! \text{ X }$

$1!)\{(0.65)^9 (0.35)^1\} + 10! / (10! \text{ X } 0!)\{(0.65)^{10} (0.35)^0\} = 0.515$

(ii) $P(x = 4) = {}^{10}C_4 p^4(1-p)^{10-4} = {}^{10}C_4(0.65)^4(0.35)^6$

$= \{10! / 4!(10 – 4)!\} (0.65)^4(0.35)^6 =$

$= 10 \text{ x } 9 \text{ x } 8 \text{ x } 7 \text{ x } 6! / 4 \text{ x } 3 \text{ x } 2 \text{ x } 6! (0.65)^4(0.35)^6$

$= 210 \{(0.65)^4(0.35)^6\} = 210 \text{ x } 0.1785 \text{ x } 0.001838 = 0.06889$

Poisson Distribution

A random variable ‘X’ is said to have Poisson distribution if its probability mass function is given by –

$$P(x) = \begin{cases} e^{-\lambda}(\lambda^{x} / x!), \; x = 0, 1, 2, \ldots\ldots \\ 0, \text{ elsewhere} \end{cases}$$

Poisson distribution is the discrete probability distribution of a discrete random variable X, which has no upper bound. It is defined for non negative values of x.

Poisson distribution is suitable for rare events for which the probability of occurrence ‘p’ is very small and the number of trials ‘n’ is very large.

Also, Binomial distribution can be approximated by Poisson distribution when $n \rightarrow \infty$ and $p \rightarrow 0$ such that $\lambda = np =$ constant.

Mean & variance of Poisson Distribution

For Poisson Distribution	For Binomial Distribution
Mean = E(X) = λ	Mean = np
Variance = V(X) = λ	Variance = V(X) = npq
Mean = Variance = λ	Mean > Variance
SD = sqrt (variance) = sqrt (λ)	SD = sqrt (variance) = sqrt (npq)

Ex 7 **:** If the probability of a bad reaction from a certain injection is 0.001. Determine the chance that out of 2000 individuals more than two will get a bad reaction.

Soln **:** Given $n = 2000$; $p = 0.001$

Mean $\lambda = np = 2000 \times 0.001 = 2$

We know, $e^{-\lambda}(\lambda^x / x!)$, $x = 0, 1, 2, \ldots\ldots$

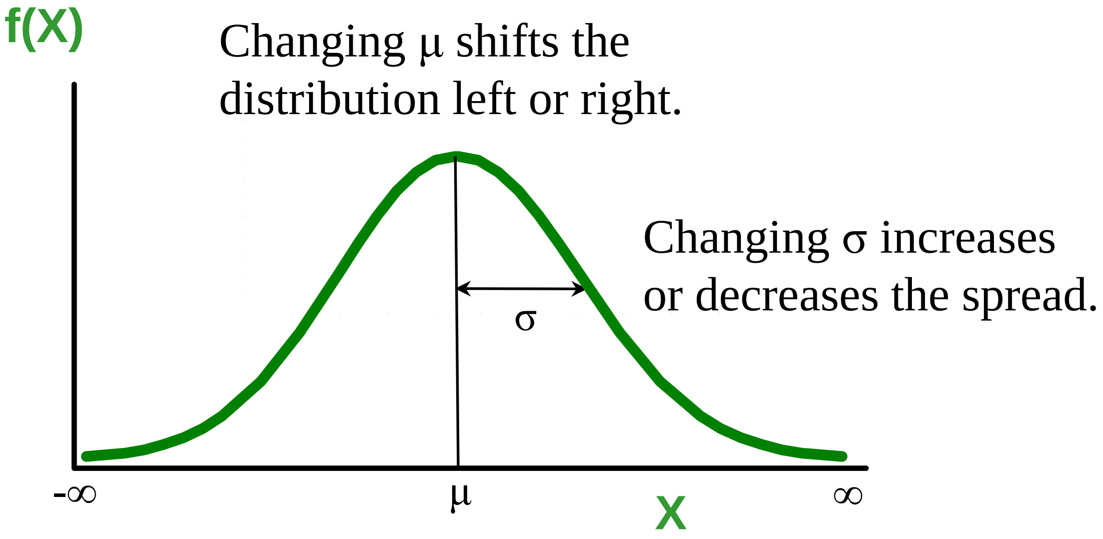
Now, find $P(x > 2) = 1 - P(x \leq 2) = 1 - \{P(0) + P(1) + P(2)\}$

$$= 1 - \{e^{-\lambda}(\lambda^0 / 0!) + e^{-\lambda}(\lambda^1 / 1!) + e^{-\lambda}(\lambda^2 / 2!)\}$$

$$= 1 - e^{-\lambda}\{1 + \lambda^1 + \lambda^2 / 2\} = 1 - e^{-2}\{1 + 2 + 2^2 / 2\}$$

$$= 1 - e^{-2}(5) = 1 - 0.1353 \times 5 = 1 - 0.6765 = 0.3235$$

The Normal Distribution : The Normal Probability Density is one of the special probability densities, usually referred to simply as the Normal Distribution (the words density & distribution are often used interchangeably in the literature of applied statistics). The equation of the normal probability density, whose graph (shaped like the cross section of a bell with different centers & spreads depending on μ and σ).



Changing μ shifts the distribution left or right.

Changing σ increases or decreases the spread.

Properties:

1. Normal curve is bell shaped and symmetric about the mean.
2. Mean = Mode = Median.
3. Total area under normal curve is equal to 1.
4. Normal curve approaches but never touches the x axis as it extends farther and farther away from the mean.

Ex 8 **:** The average score of students in a statistics test was 78 with a standard deviation of 8. If the test scores are normally distributed, find the probability that a student receives a test score – (i) less than 90, (ii) More than 90 and (iii) between 80 and 96.

Soln **:** Given $\mu = 78$ and $\sigma = 8$

(i) Required to find $P(x < 90)$.

Converting it to Std Normal $P(Z < 90)$ where,

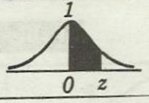
$Z = (x - \mu) / \sigma = (90 - 78) / 8 = 1.5$ ie. $P(Z < 1.5) = 0.9332$ (From Z-table)

(ii) $P(x > 90) = P(Z > 1.5) = 1 - 0.9332 = 0.0668$

(iii) $P(80 < x < 96) = P[(80 - 78) /8 < Z < (96 - 78) / 8] = P(1.5 < Z < 2.25)$

$= 0.9878 - 0.9332 = 0.0546$

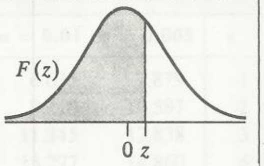
Table V : Area under the Normal curve



z	0	1	2	3	4	5	6	7	8	9
0.0	0.0000	0.0040	0.0080	0.0120	0.0160	0.0199	0.0239	0.0279	0.0319	0.0359
0.1	0.0398	0.0438	0.0478	0.0517	0.0557	0.0596	0.0636	0.0675	0.0714	0.0754
0.2	0.0793	0.0832	0.0871	0.0910	0.0948	0.0987	0.1026	0.1064	0.1103	0.1141
0.3	0.1179	0.1217	0.1255	0.1293	0.1331	0.1368	0.1406	0.1443	0.1480	0.1517
0.4	0.1554	0.1591	0.1628	0.1664	0.1700	0.1736	0.1772	0.1808	0.1844	0.1879
0.5	0.1915	0.1950	0.1985	0.2019	0.2054	0.2088	0.2123	0.2157	0.2190	0.2224
0.6	0.2258	0.2291	0.2324	0.2357	0.2389	0.2422	0.2454	0.2486	0.2518	0.2549
0.7	0.2580	0.2612	0.2642	0.2673	0.2704	0.2734	0.2764	0.2794	0.2823	0.2852
0.8	0.2881	0.2910	0.2939	0.2967	0.2996	0.3023	0.3051	0.3078	0.3106	0.3133
0.9	0.3159	0.3186	0.3212	0.3238	0.3264	0.3289	0.3315	0.3340	0.3365	0.3389
1.0	0.3413	0.3438	0.3461	0.3485	0.3508	0.3531	0.3554	0.3577	0.3599	0.3621
1.1	0.3643	0.3665	0.3686	0.3708	0.3729	0.3749	0.3770	0.3790	0.3810	0.3830
1.2	0.3849	0.3869	0.3888	0.3907	0.3925	0.3944	0.3962	0.3980	0.3997	0.4015
1.3	0.4032	0.4049	0.4066	0.4082	0.4099	0.4115	0.4131	0.4147	0.4162	0.4177
1.4	0.4192	0.4207	0.4222	0.4236	0.4251	0.4265	0.4279	0.4292	0.4306	0.4319
1.5	0.4332	0.4345	0.4357	0.4370	0.4382	0.4394	0.4406	0.4418	0.4429	0.4441
1.6	0.4452	0.4463	0.4474	0.4484	0.4495	0.4505	0.4515	0.4525	0.4535	0.4545
1.7	0.4554	0.4564	0.4573	0.4582	0.4591	0.4599	0.4608	0.4616	0.4625	0.4633
1.8	0.4641	0.4649	0.4656	0.4664	0.4671	0.4678	0.4686	0.4693	0.4699	0.4706
1.9	0.4713	0.4719	0.4726	0.4732	0.4738	0.4744	0.4750	0.4756	0.4761	0.4767
2.0	0.4772	0.4778	0.4783	0.4788	0.4793	0.4798	0.4803	0.4808	0.4812	0.4817
2.1	0.4821	0.4826	0.4830	0.4834	0.4838	0.4842	0.4846	0.4850	0.4854	0.4857
2.2	0.4861	0.4864	0.4868	0.4871	0.4875	0.4878	0.4881	0.4884	0.4887	0.4890
2.3	0.4893	0.4896	0.4898	0.4901	0.4904	0.4906	0.4909	0.4911	0.4913	0.4916
2.4	0.4918	0.4920	0.4922	0.4925	0.4927	0.4929	0.4931	0.4932	0.4934	0.4936
2.5	0.4938	0.4940	0.4941	0.4943	0.4945	0.4946	0.4948	0.4949	0.4951	0.4952
2.6	0.4953	0.4955	0.4956	0.4957	0.4959	0.4960	0.4961	0.4962	0.4963	0.4964
2.7	0.4965	0.4966	0.4967	0.4968	0.4969	0.4970	0.4971	0.4972	0.4973	0.4974
2.8	0.4974	0.4975	0.4976	0.4977	0.4977	0.4978	0.4979	0.4979	0.4980	0.4981
2.9	0.4981	0.4982	0.4982	0.4983	0.4984	0.4984	0.4985	0.4985	0.4986	0.4986
3.0	0.4987	0.4987	0.4987	0.4988	0.4988	0.4989	0.4989	0.4989	0.4990	0.4990
3.1	0.4990	0.4991	0.4991	0.4991	0.4992	0.4992	0.4992	0.4992	0.4993	0.4993

Table 3 *(continued from page 514)*

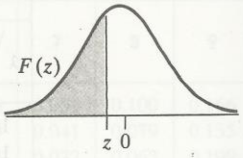
$$F(z) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} e^{-t^2/2}\,dt$$



z	0.00	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09
0.0	0.5000	0.5040	0.5080	0.5120	0.5160	0.5199	0.5239	0.5279	0.5319	0.5359
0.1	0.5398	0.5438	0.5478	0.5517	0.5557	0.5596	0.5636	0.5675	0.5714	0.5753
0.2	0.5973	0.5832	0.5871	0.5910	0.5948	0.5987	0.6026	0.6064	0.6103	0.6141
0.3	0.6179	0.6217	0.6255	0.6293	0.6331	0.6368	0.6406	0.6443	0.6480	0.6517
0.4	0.6554	0.6591	0.6628	0.6664	0.6700	0.6736	0.6772	0.6808	0.6844	0.6879
0.5	0.6915	0.6950	0.6985	0.7019	0.7054	0.7088	0.7123	0.7157	0.7190	0.7224
0.6	0.7257	0.7291	0.7324	0.7357	0.7389	0.7422	0.7454	0.7486	0.7517	0.7549
0.7	0.7580	0.7611	0.7642	0.7673	0.7704	0.7734	0.7764	0.7794	0.7823	0.7852
0.8	0.7881	0.7910	0.7939	0.7967	0.7995	0.8023	0.8051	0.8078	0.8106	0.8133
0.9	0.8159	0.8186	0.8212	0.8238	0.8264	0.8289	0.8315	0.8340	0.8365	0.8389
1.0	0.8413	0.8438	0.8461	0.8485	0.8508	0.8531	0.8554	0.8577	0.8599	0.8621
1.1	0.8643	0.8665	0.8686	0.8708	0.8729	0.8749	0.8770	0.8790	0.8810	0.8830
1.2	0.8849	0.8869	0.8888	0.8907	0.8925	0.8944	0.8962	0.8980	0.8997	0.9015
1.3	0.9032	0.9049	0.9066	0.9082	0.9099	0.9115	0.9131	0.9147	0.9162	0.9177
1.4	0.9192	0.9207	0.9222	0.9236	0.9251	0.9265	0.9279	0.9292	0.9306	0.9319
1.5	0.9332	0.9345	0.9357	0.9370	0.9382	0.9394	0.9406	0.9418	0.9429	0.9441
1.6	0.9452	0.9463	0.9474	0.9484	0.9495	0.9505	0.9515	0.9525	0.9535	0.9545
1.7	0.9554	0.9564	0.9573	0.9582	0.9591	0.9599	0.9608	0.9616	0.9625	0.9633
1.8	0.9641	0.9649	0.9656	0.9664	0.9671	0.9678	0.9686	0.9693	0.9699	0.9706
1.9	0.9713	0.9719	0.9726	0.9732	0.9738	0.9744	0.9750	0.9756	0.9761	0.9767
2.0	0.9772	0.9778	0.9783	0.9788	0.9793	0.9798	0.9803	0.9808	0.9812	0.9817
2.1	0.9821	0.9826	0.9830	0.9834	0.9838	0.9842	0.9846	0.9850	0.9854	0.9857
2.2	0.9861	0.9864	0.9868	0.9871	0.9875	0.9878	0.9881	0.9884	0.9887	0.9890
2.3	0.9893	0.9896	0.9898	0.9901	0.9904	0.9906	0.9909	0.9911	0.9913	0.9916
2.4	0.9918	0.9920	0.9922	0.9925	0.9927	0.9929	0.9931	0.9932	0.9934	0.9936
2.5	0.9938	0.9940	0.9941	0.9943	0.9945	0.9946	0.9948	0.9949	0.9951	0.9952
2.6	0.9953	0.9955	0.9956	0.9957	0.9959	0.9960	0.9961	0.9962	0.9963	0.9964
2.7	0.9965	0.9966	0.9967	0.9968	0.9969	0.9970	0.9971	0.9972	0.9973	0.9974
2.8	0.9974	0.9975	0.9976	0.9977	0.9977	0.9978	0.9979	0.9979	0.9980	0.9981
2.9	0.9981	0.9982	0.9982	0.9983	0.9984	0.9984	0.9985	0.9985	0.9986	0.9986
3.0	0.9987	0.9987	0.9987	0.9988	0.9988	0.9989	0.9989	0.9989	0.9990	0.9990
3.1	0.9990	0.9991	0.9991	0.9991	0.9992	0.9992	0.9992	0.9992	0.9993	0.9993
3.2	0.9993	0.9993	0.9994	0.9994	0.9994	0.9994	0.9994	0.9995	0.9995	0.9995
3.3	0.9995	0.9995	0.9995	0.9996	0.9996	0.9996	0.9996	0.9996	0.9996	0.9997
3.4	0.9997	0.9997	0.9997	0.9997	0.9997	0.9997	0.9997	0.9997	0.9997	0.9998
3.5	0.9998									
4.0	0.99997									
5.0	0.9999997									

Table 3 Standard Normal Distribution Function

$$F(z) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} e^{-t^2/2}\,dt$$



z	0.00	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09
−5.0	0.0000003									
−4.0	0.00003									
−3.5	0.0002									
−3.4	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0002
−3.3	0.0005	0.0005	0.0005	0.0004	0.0004	0.0004	0.0004	0.0004	0.0006	0.0003
−3.2	0.0007	0.0007	0.0006	0.0006	0.0006	0.0006	0.0006	0.0005	0.0005	0.0005
−3.1	0.0010	0.0009	0.0009	0.0009	0.0008	0.0008	0.0008	0.0008	0.0007	0.0007
−3.0	0.0013	0.0013	0.0013	0.0012	0.0012	0.0011	0.0011	0.0011	0.0010	0.0010
−2.9	0.0019	0.0018	0.0018	0.0017	0.0016	0.0016	0.0015	0.0015	0.0014	0.0014
−2.8	0.0026	0.0025	0.0024	0.0023	0.0023	0.0022	0.0021	0.0021	0.0020	0.0019
−2.7	0.0035	0.0034	0.0033	0.0032	0.0031	0.0030	0.0029	0.0028	0.0027	0.0026
−2.6	0.0047	0.0045	0.0044	0.0043	0.0041	0.0040	0.0039	0.0038	0.0037	0.0036
−2.5	0.0062	0.0060	0.0059	0.0057	0.0055	0.0054	0.0052	0.0051	0.0049	0.0048
−2.4	0.0082	0.0080	0.0078	0.0075	0.0073	0.0071	0.0069	0.0068	0.0066	0.0064
−2.3	0.0107	0.0104	0.0102	0.0099	0.0096	0.0094	0.0091	0.0089	0.0087	0.0084
−2.2	0.0139	0.0136	0.0132	0.0129	0.0125	0.0122	0.0119	0.0116	0.0113	0.0110
−2.1	0.0179	0.0174	0.0170	0.0166	0.0162	0.0158	0.0154	0.0150	0.0146	0.0143
−2.0	0.0228	0.0222	0.0217	0.0212	0.0207	0.0202	0.0197	0.0192	0.0188	0.0183
−1.9	0.0287	0.0281	0.0274	0.0268	0.0262	0.0256	0.0250	0.0244	0.0239	0.0233
−1.8	0.0359	0.0351	0.0344	0.0336	0.0329	0.0322	0.0314	0.0307	0.0301	0.0294
−1.7	0.0446	0.0436	0.0427	0.0418	0.0409	0.0401	0.0392	0.0384	0.0375	0.0367
−1.6	0.0548	0.0537	0.0526	0.0516	0.0505	0.0495	0.0485	0.0475	0.0465	0.0455
−1.5	0.0668	0.0655	0.0643	0.0630	0.0618	0.0606	0.0594	0.0582	0.0571	0.0559
−1.4	0.0808	0.0793	0.0778	0.0764	0.0749	0.0735	0.0721	0.0708	0.0694	0.0681
−1.3	0.0968	0.0951	0.0934	0.0918	0.0901	0.0885	0.0869	0.0853	0.0838	0.0823
−1.2	0.1151	0.1131	0.1112	0.1093	0.1075	0.1056	0.1038	0.1020	0.1003	0.0985
−1.1	0.1357	0.1335	0.1314	0.1292	0.1271	0.1251	0.1230	0.1210	0.1190	0.1170
−1.0	0.1587	0.1562	0.1539	0.1515	0.1492	0.1469	0.1446	0.1423	0.1401	0.1379
−0.9	0.1841	0.1814	0.1788	0.1762	0.1736	0.1711	0.1685	0.1660	0.1635	0.1611
−0.8	0.2119	0.2090	0.2061	0.2033	0.2005	0.1977	0.1949	0.1922	0.1894	0.1867
−0.7	0.2420	0.2389	0.2358	0.2327	0.2296	0.2266	0.2236	0.2206	0.2177	0.2148
−0.6	0.2743	0.2709	0.2676	0.2643	0.2611	0.2578	0.2546	0.2514	0.2483	0.2451
−0.5	0.3085	0.3050	0.3015	0.2981	0.2946	0.2912	0.2877	0.2843	0.2810	0.2776
−0.4	0.3446	0.3409	0.3372	0.3336	0.3300	0.3264	0.3228	0.3192	0.3156	0.3121
−0.3	0.3821	0.3783	0.3745	0.3707	0.3669	0.3632	0.3594	0.3557	0.3520	0.3483
−0.2	0.4207	0.4168	0.4129	0.4090	0.4052	0.4013	0.3974	0.3936	0.3897	0.3859
−0.1	0.4602	0.4562	0.4522	0.4483	0.4443	0.4404	0.4364	0.4325	0.4286	0.4247
−0.0	0.5000	0.4960	0.4920	0.4880	0.4840	0.4801	0.4761	0.4721	0.4681	0.4641

(continued on following page)

Sampling

Sampling is a method or process of selecting samples from populations.

Data are gathered from samples and conclusions are drawn about the population as a part of the inferential statistics process.

A sample provides a reasonable means for gathering useful decision-making information that might be otherwise unattainable and unaffordable.

Sampling Distribution-Z distribution

Sampling distribution is a set of collection of all calculated values of a statistic where each value is obtained from each sample of a total possible random samples each of size n that can be taken from a population. The sampling distribution follows probability distribution.

Central Limit Theorem (CLT) : If $\bar{x}$ is the mean of a sample size n taken from a population having the mean μ and the finite variance σ^2, then $Z = \dfrac{\bar{x} - \mu}{\sigma/\mathrm{sqrt}(n)}$ is a random variable whose distribution approaches to standard normal distribution as $n \to \infty$.

Note : Central limit theorem is better applicable if the sample size $n \geq 25$ but in general for $n \geq 30$.

Ex 9 **:** The time at the counter for a customer to be served at a post office can be modelled as a random variable having mean 176 seconds and variance 256 (seconds)^2. The sample mean $\bar{x}$ will be obtained from the times for a random sample of 100 customers. What is the probability that $\bar{x}$ will be between 175 and 178 seconds?

Soln **:** Given $\mu = 176$, $\sigma^2 = 256$ implies $\sigma = 16$, n = 100 (> 30)

Then by using CLT, $Z = \frac{\bar{x} - \mu}{\sigma/\text{sqrt}(n)} = \frac{\bar{x} - 176}{16/\text{sqrt}(100)} = \frac{\bar{x} - 176}{1.6}$ is a standard normal variable.

The probability that $\bar{x}$ will be between 175 and 178 seconds is

$P(175 < \bar{x} < 178) = P(\frac{175 - 176}{1.6} < Z < \frac{178 - 176}{1.6}) =$

P(-0.625 < Z < 1.25) = F(1.25) - F(-0.625) = 0.8944 - 0.2659 (Taken from the Z - table values of standard normal distribution) = 0.628

Exact Sampling Distributions (Small Sampling Distributions)

t- Distribution **:** Let $(x_1, x_2, \ldots, x_n)$ be a small sample drawn at random from a normal population with mean μ and un-known variance σ^2.

Let $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ be the sample mean which is the estimator of population mean μ and $S^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$ be the sample variance which is the estimator of population variance σ^2.

Then the distribution of $\bar{x}$ has the mean $E(\bar{x}) = \mu$ and variance $V(\bar{x}) = \frac{\boldsymbol{\sigma^2}}{n} = \frac{S^2}{n}$ (since, σ^2 is not known then it is replaced by its estimator S^2) and $SE(\bar{x}) = \sqrt{V(\bar{x})} = S/\sqrt{n}$.

Then the statistic $t = \frac{\bar{x} - \mu}{S/\sqrt{n}}$ is a random variable follows Student's t distribution with ϑ(nue) =(n-1) degree of freedom (parameter)where $-\infty < t < \infty$.

t distribution is symmetric distribution. It is symmetric at mean 0 and variance greater than unity (= ϑ / (ϑ -2)). t distribution behaves similar to standard normal distribution.

(Degrees of freedom $\vartheta = n - k$ is the difference between ‘n’ the sample size and k is the number of population parameters which are calculated using the sample data).

The t-distribution curve is symmetric about the mean zero, unimodal, bell shaped and asymptotic on both sides of t-axis.

Ex 10 **:** A random sample of size 25 from a normal population has the mean
$\bar{x} = 47.5$ and the standard deviation S = 8.4. Does this information tend to support or refute the claim that the mean of the population μ = 42.1?

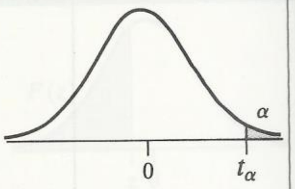
Soln **:** Given n = 25, $\bar{x} = 47.5$, S = 8.4, μ = 42.1

Then $t_{(n-1)df} = \frac{\bar{x}-\mu}{S/\sqrt{n}} \Rightarrow t_{24df} = \frac{47.5-42.1}{8.4/\sqrt{25}} = 5.4/1.68 = 3.21$

From Table of t distribution we observe,
$P(t_{24df} > 2.797) = 0.005$ which is the area of t_{24df} between 2.797 and ∞.
Then $P(t_{24df} > 3.21) < 0.005$ since the area of t_{24df} between 3.21 and ∞ is less than that of the area between 2.797 and ∞ as it is known that
$2.797 < 3.21 < \infty$.

Therefore, $P(t_{24df} > 3.21) < 0.005$ tends to refute (reject) the claim that the mean of the population μ = 42.1

Table 4 Values of t_α



ν	$\alpha = 0.10$	$\alpha = 0.05$	$\alpha = 0.025$	$\alpha = 0.01$	$\alpha = 0.00833$	$\alpha = 0.00625$	$\alpha = 0.005$	ν
1	3.078	6.314	12.706	31.821	38.204	50.923	63.657	1
2	1.886	2.920	4.303	6.965	7.650	8.860	9.925	2
3	1.638	2.353	3.182	4.541	4.857	5.392	5.841	3
4	1.533	2.132	2.776	3.747	3.961	4.315	4.604	4
5	1.476	2.015	2.571	3.365	3.534	3.810	4.032	5
6	1.440	1.943	2.447	3.143	3.288	3.521	3.707	6
7	1.415	1.895	2.365	2.998	3.128	3.335	3.499	7
8	1.397	1.860	2.306	2.896	3.016	3.206	3.355	8
9	1.383	1.833	2.262	2.821	2.934	3.111	3.250	9
10	1.372	1.812	2.228	2.764	2.870	3.038	3.169	10
11	1.363	1.796	2.201	2.718	2.820	2.891	3.106	11
12	1.356	1.782	2.179	2.681	2.780	2.934	3.055	12
13	1.350	1.771	2.160	2.650	2.746	2.896	3.012	13
14	1.345	1.761	2.145	2.624	2.718	2.864	2.977	14
15	1.341	1.753	2.131	2.602	2.694	2.837	2.947	15
16	1.337	1.746	2.120	2.583	2.673	2.813	2.921	16
17	1.333	1.740	2.110	2.567	2.655	2.793	2.898	17
18	1.330	1.734	2.101	2.552	2.639	2.775	2.878	18
19	1.328	1.729	2.093	2.539	2.625	2.759	2.861	19
20	1.325	1.725	2.086	2.528	2.613	2.744	2.845	20
21	1.323	1.721	2.080	2.518	2.602	2.732	2.831	21
22	1.321	1.717	2.074	2.508	2.591	2.720	2.819	22
23	1.319	1.714	2.069	2.500	2.582	2.710	2.807	23
24	1.318	1.711	2.064	2.492	2.574	2.700	2.797	24
25	1.316	1.708	2.060	2.485	2.566	2.692	2.787	25
26	1.315	1.706	2.056	2.479	2.559	2.684	2.779	26
27	1.314	1.703	2.052	2.473	2.553	2.676	2.771	27
28	1.313	1.701	2.048	2.467	2.547	2.669	2.763	28
29	1.311	1.699	2.045	2.462	2.541	2.663	2.756	29
inf.	1.282	1.645	1.960	2.326	2.394	2.498	2.576	inf.

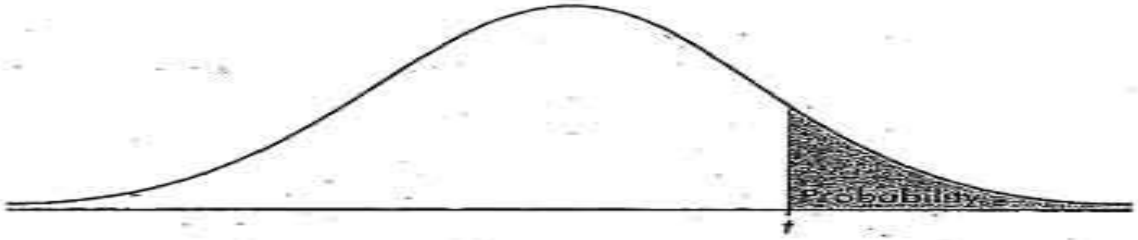


TABLE B: t-DISTRIBUTION CRITICAL VALUES

	Tail probability p											
df	.25	.20	.15	.10	.05	.025	.02	.01	.005	.0025	.001	.0005
1	1.000	1.376	1.963	3.078	6.314	12.71	15.89	31.82	63.66	127.3	318.3	636.6
2	.816	1.061	1.386	1.886	2.920	4.303	4.849	6.965	9.925	14.09	22.33	31.60
3	.765	.978	1.250	1.638	2.353	3.182	3.482	4.541	5.841	7.453	10.21	12.92
4	.741	.941	1.190	1.533	2.132	2.776	2.999	3.747	4.604	5.598	7.173	8.610
5	.727	.920	1.156	1.476	2.015	2.571	2.757	3.365	4.032	4.773	5.893	6.869
6	.718	.906	1.134	1.440	1.943	2.447	2.612	3.143	3.707	4.317	5.208	5.959
7	.711	.896	1.119	1.415	1.895	2.365	2.517	2.998	3.499	4.029	4.785	5.408
8	.706	.889	1.108	1.397	1.860	2.306	2.449	2.896	3.355	3.833	4.501	5.041
9	.703	.883	1.100	1.383	1.833	2.262	2.398	2.821	3.250	3.690	4.297	4.781
10	.700	.879	1.093	1.372	1.812	2.228	2.359	2.764	3.169	3.581	4.144	4.587
11	.697	.876	1.088	1.363	1.796	2.201	2.328	2.718	3.106	3.497	4.025	4.437
12	.695	.873	1.083	1.356	1.782	2.179	2.303	2.681	3.055	3.428	3.930	4.318
13	.694	.870	1.079	1.350	1.771	2.160	2.282	2.650	3.012	3.372	3.852	4.221
14	.692	.868	1.076	1.345	1.761	2.145	2.264	2.624	2.977	3.326	3.787	4.140
15	.691	.866	1.074	1.341	1.753	2.131	2.249	2.602	2.947	3.286	3.733	4.073
16	.690	.865	1.071	1.337	1.746	2.120	2.235	2.583	2.921	3.252	3.686	4.015
17	.689	.863	1.069	1.333	1.740	2.110	2.224	2.567	2.898	3.222	3.646	3.965
18	.688	.862	1.067	1.330	1.734	2.101	2.214	2.552	2.878	3.197	3.611	3.922
19	.688	.861	1.066	1.328	1.729	2.093	2.205	2.539	2.861	3.174	3.579	3.883
20	.687	.860	1.064	1.325	1.725	2.086	2.197	2.528	2.845	3.153	3.552	3.850
21	.686	.859	1.063	1.323	1.721	2.080	2.189	2.518	2.831	3.135	3.527	3.819
22	.686	.858	1.061	1.321	1.717	2.074	2.183	2.508	2.819	3.119	3.505	3.792
23	.685	.858	1.060	1.319	1.714	2.069	2.177	2.500	2.807	3.104	3.485	3.768
24	.685	.857	1.059	1.318	1.711	2.064	2.172	2.492	2.797	3.091	3.467	3.745
25	.684	.856	1.058	1.316	1.708	2.060	2.167	2.485	2.787	3.078	3.450	3.725
26	.684	.856	1.058	1.315	1.706	2.056	2.162	2.479	2.779	3.067	3.435	3.707
27	.684	.855	1.057	1.314	1.703	2.052	2.158	2.473	2.771	3.057	3.421	3.690
28	.683	.855	1.056	1.313	1.701	2.048	2.154	2.467	2.763	3.047	3.408	3.674
29	.683	.854	1.055	1.311	1.699	2.045	2.150	2.462	2.756	3.038	3.396	3.659
30	.683	.854	1.055	1.310	1.697	2.042	2.147	2.457	2.750	3.030	3.385	3.646
40	.681	.851	1.050	1.303	1.684	2.021	2.123	2.423	2.704	2.971	3.307	3.551
50	.679	.849	1.047	1.299	1.676	2.009	2.109	2.403	2.678	2.937	3.261	3.496
60	.679	.848	1.045	1.296	1.671	2.000	2.099	2.390	2.660	2.915	3.232	3.460
80	.678	.846	1.043	1.292	1.664	1.990	2.088	2.374	2.639	2.887	3.195	3.416
100	.677	.845	1.042	1.290	1.660	1.984	2.081	2.364	2.626	2.871	3.174	3.390
1000	.675	.842	1.037	1.282	1.646	1.962	2.056	2.330	2.581	2.813	3.098	3.300
∞	.674	.841	1.036	1.282	1.645	1.960	2.054	2.326	2.576	2.807	3.091	3.291
	50%	60%	70%	80%	90%	95%	96%	98%	99%	99.5%	99.8%	99.9%
	Confidence level C											

F distribution

Let $(x_1, x_2, \ldots, x_{n1})$ be a small sample drawn at random from a normal population with mean μ_1 (un-known) and variance $\sigma^2{}_1$.

Let $\bar{x} = \frac{1}{n_1}\sum_{i=1}^{n_1} x_i$ be the sample mean which is the estimator of population mean μ_1 and $S^2{}_1 = \frac{1}{n_1-1}\sum_{i=1}^{n_1}(x_i - \bar{x})^2$ be the sample variance which is the estimator of population variance $\sigma^2{}_1$.

Let $(y_1, y_2, \ldots, y_{n2})$ be a small sample drawn at random from another independent normal population with mean μ_2 (un-known) and variance $\sigma^2{}_2$.

Let $\bar{y} = \frac{1}{n_2}\sum_{i=1}^{n_2} y_i$ be the sample mean which is the estimator of population mean μ_2 and $S^2{}_2 = \frac{1}{n_2-1}\sum_{i=1}^{n_2}(y_i - \bar{y})^2$ be the sample variance which is the estimator of population variance $\sigma^2{}_2$.

Also assume $\sigma^2{}_1 = \sigma^2{}_2 = \sigma^2$ (unknown).

A Random variable having F-Distribution – Theorem : If S^2_1 and S^2_2 are the variances of independent random samples of size n_1 and n_2 respectively, taken from two normal populations having the same variance, then -

$F = S^2_1 / S^2_2$

is a random variable having F-Distribution with parameters $\upsilon_1 = n_1 - 1$ and $\upsilon_2 = n_2 - 1$.

F Values for $\alpha = 0{:}05$

d_1

d_2	1	2	3	4	5	6	7	8	9
1	161.4	199.5	215.7	224.6	230.2	234.0	236.8	238.9	240.5
2	18.51	19.00	19.16	19.25	19.3	19.33	19.35	19.37	19.38
3	10.13	9.55	9. 28	9.12	9.01	8.94	8.89	8.85	8.81
4	7.71	6.94	6.59	6.39	6.26	6.16	6.09	6.04	6.00
5	6.61	5.79	5.41	5.19	5.05	4.95	4.88	4.82	4.77
6	5.99	5.14	4.76	4.53	4.39	4.28	4.21	4.15	4.10
7	5.59	4.74	4.35	4.12	3.97	3.87	3.79	3.73	3.68
8	5.32	4.46	4.07	3.84	3.69	3.58	3.50	3.44	3.39

9	5.12	4.26	3.86	3.63	3.48	3.37	3.29	3.23	3.18
10	4.96	4.10	3.71	3.48	3.33	3.22	3.14	3.07	3.02
11	4.84	3.98	3.59	3.36	3.20	3.09	3.01	2.95	2.90
12	4.75	3.89	3.49	3.26	3.11	3.00	2.91	2.85	2.80
13	4.67	3.81	3.41	3.18	3.03	2.92	2.83	2.77	2.71
14	4.60	3.74	3.34	3.11	2.96	2.85	2.76	2.70	2.65
15	4.54	3.68	3.29	3.06	2.90	2.79	2.71	2.64	2.59
16	4.49	3.63	3.24	3.01	2.85	2.74	2.66	2.59	2.54
17	4.45	3.59	3.20	2.96	2.81	2.70	2.61	2.55	2.49
18	4.41	3.55	3.16	2.93	2.77	2.66	2.58	2.51	2.46
19	4.38	3.52	3.13	2.90	2.74	2.63	2.54	2.48	2.42
20	4.35	3.49	3.10	2.87	2.71	2.60	2.51	2.45	2.39
21	4.32	3.47	3.07	2.84	2.68	2.57	2.49	2.42	2.37
22	4.30	3.44	3.05	2.82	2.66	2.55	2.46	2.40	2.34
23	4.28	3.42	3.03	2.80	2.64	2.53	2.44	2.37	2.32
24	4.26	3.40	3.01	2.78	2.62	2.51	2.42	2.36	2.30
25	4.24	3.39	2.99	2.76	2.60	2.49	2.40	2.34	2.28
26	4.23	3.37	2.98	2.74	2.59	2.47	2.39	2.32	2.27
27	4.21	3.35	2.96	2.73	2.57	2.46	2.37	2.31	2.25
28	4.20	3.34	2.95	2.71	2.56	2.45	2.36	2.29	2.24
29	4.18	3.33	2.93	2.70	2.55	2.43	2.35	2.28	2.22
30	4.17	3.32	2.92	2.69	2.53	2.42	2.33	2.27	2.21
40	4.08	3.23	2.84	2.61	2.45	2.34	2.25	2.18	2.12
60	4.00	3.15	2.76	2.53	2.37	2.25	2.17	2.10	2.04
120	3.92	3.07	2.68	2.45	2.29	2.17	2.09	2.02	1.96
inf	3.84	3.00	2.60	2.37	2.21	2.10	2.01	1.94	1.88

For F-Distribution F = $S^2{}_1 / S^2{}_2$ with $[(n_1 - 1), (n_2 - 1)]$ degrees of freedom.

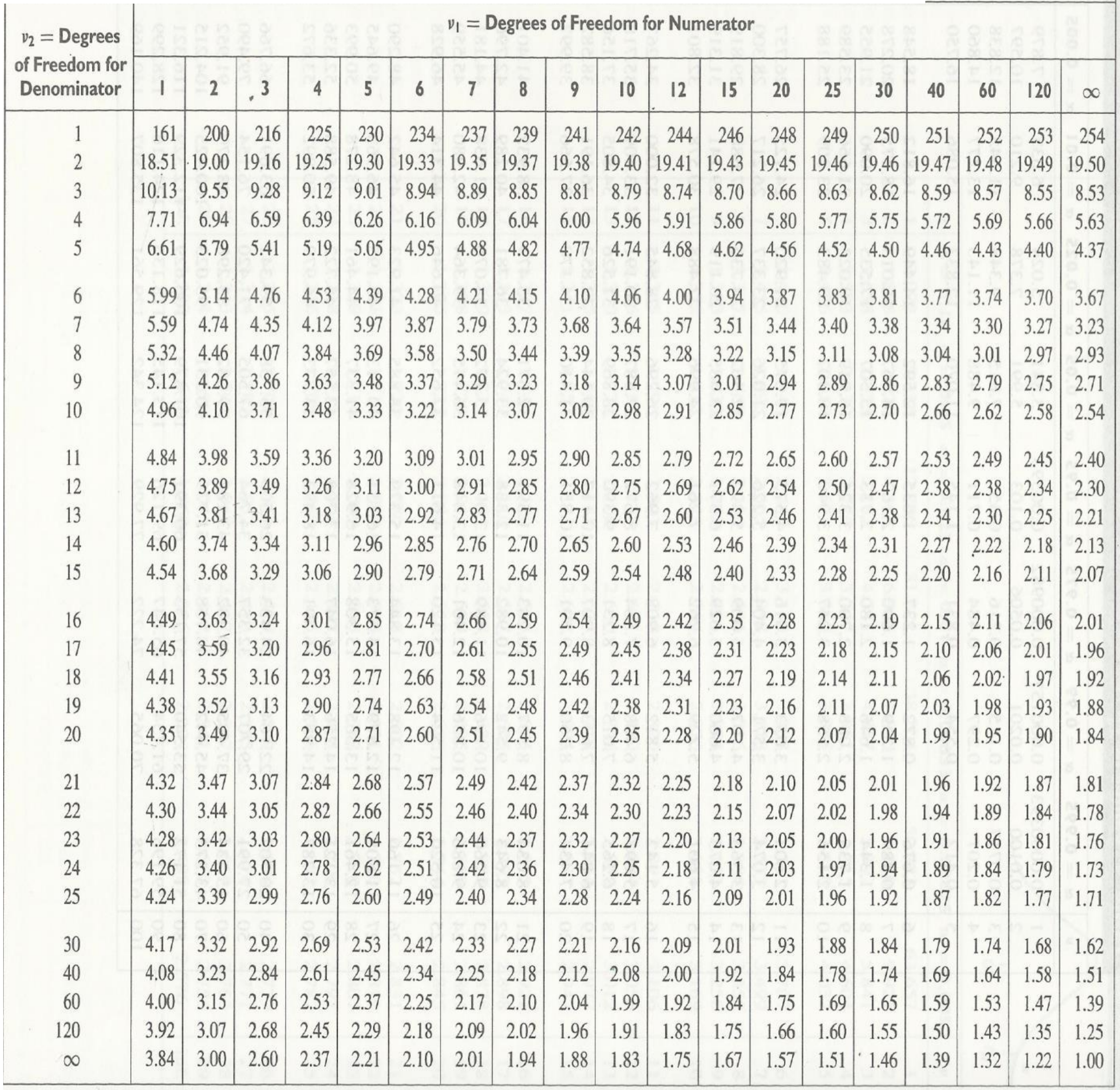
Ex : If two independent random samples of size $n_1 = 7$ and $n_2 = 13$ are taken from a normal population. What is the probability that the variance of the first sample will be at least 3 times as large as that of the second sample?

Soln : To find $P(S^2{}_1 \geq 3\, S^2{}_2) = P(S^2{}_1 / S^2{}_2 \geq 3)$

$= P(F \geq 3) = P(F([(n_1 - 1), (n_2 - 1)] \geq 3) =$

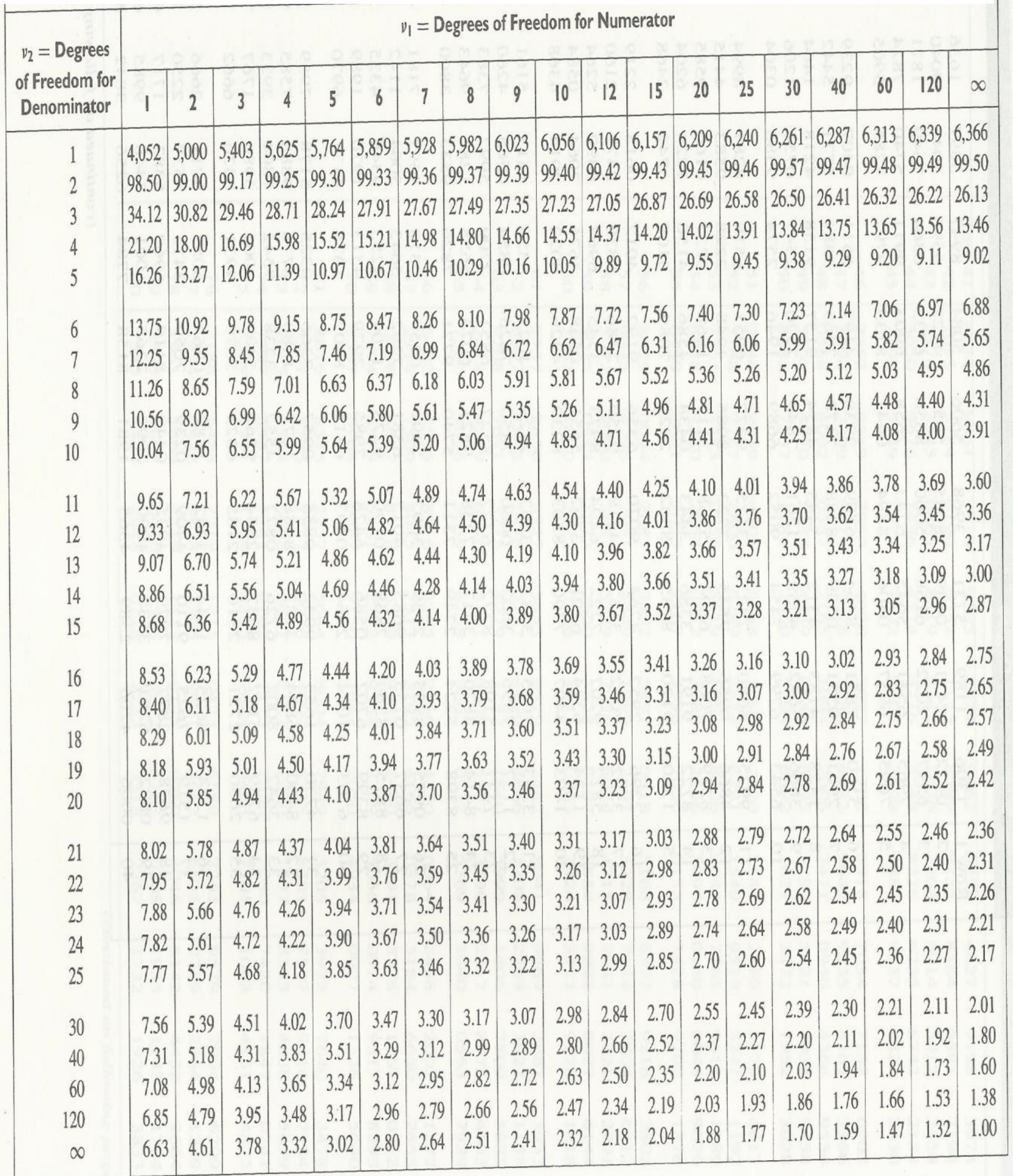
$P(F(6,12) \geq 3) = 0.05$ observed from F tables.

Values of $F_{0.05}$ F-table



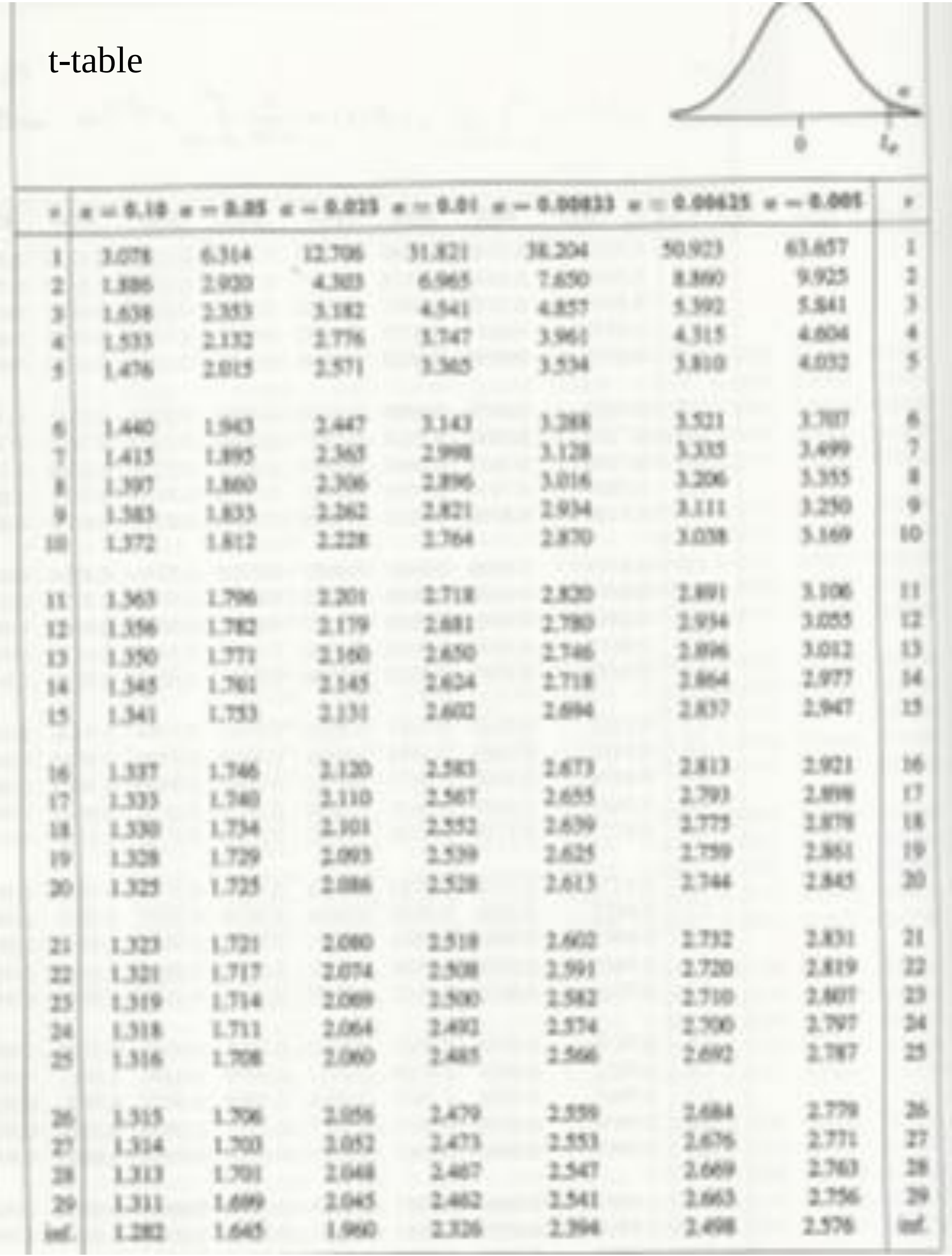
ν_2 = Degrees of Freedom for Denominator	ν_1 = Degrees of Freedom for Numerator																		
	1	2	3	4	5	6	7	8	9	10	12	15	20	25	30	40	60	120	∞
1	161	200	216	225	230	234	237	239	241	242	244	246	248	249	250	251	252	253	254
2	18.51	19.00	19.16	19.25	19.30	19.33	19.35	19.37	19.38	19.40	19.41	19.43	19.45	19.46	19.46	19.47	19.48	19.49	19.50
3	10.13	9.55	9.28	9.12	9.01	8.94	8.89	8.85	8.81	8.79	8.74	8.70	8.66	8.63	8.62	8.59	8.57	8.55	8.53
4	7.71	6.94	6.59	6.39	6.26	6.16	6.09	6.04	6.00	5.96	5.91	5.86	5.80	5.77	5.75	5.72	5.69	5.66	5.63
5	6.61	5.79	5.41	5.19	5.05	4.95	4.88	4.82	4.77	4.74	4.68	4.62	4.56	4.52	4.50	4.46	4.43	4.40	4.37
6	5.99	5.14	4.76	4.53	4.39	4.28	4.21	4.15	4.10	4.06	4.00	3.94	3.87	3.83	3.81	3.77	3.74	3.70	3.67
7	5.59	4.74	4.35	4.12	3.97	3.87	3.79	3.73	3.68	3.64	3.57	3.51	3.44	3.40	3.38	3.34	3.30	3.27	3.23
8	5.32	4.46	4.07	3.84	3.69	3.58	3.50	3.44	3.39	3.35	3.28	3.22	3.15	3.11	3.08	3.04	3.01	2.97	2.93
9	5.12	4.26	3.86	3.63	3.48	3.37	3.29	3.23	3.18	3.14	3.07	3.01	2.94	2.89	2.86	2.83	2.79	2.75	2.71
10	4.96	4.10	3.71	3.48	3.33	3.22	3.14	3.07	3.02	2.98	2.91	2.85	2.77	2.73	2.70	2.66	2.62	2.58	2.54
11	4.84	3.98	3.59	3.36	3.20	3.09	3.01	2.95	2.90	2.85	2.79	2.72	2.65	2.60	2.57	2.53	2.49	2.45	2.40
12	4.75	3.89	3.49	3.26	3.11	3.00	2.91	2.85	2.80	2.75	2.69	2.62	2.54	2.50	2.47	2.38	2.38	2.34	2.30
13	4.67	3.81	3.41	3.18	3.03	2.92	2.83	2.77	2.71	2.67	2.60	2.53	2.46	2.41	2.38	2.34	2.30	2.25	2.21
14	4.60	3.74	3.34	3.11	2.96	2.85	2.76	2.70	2.65	2.60	2.53	2.46	2.39	2.34	2.31	2.27	2.22	2.18	2.13
15	4.54	3.68	3.29	3.06	2.90	2.79	2.71	2.64	2.59	2.54	2.48	2.40	2.33	2.28	2.25	2.20	2.16	2.11	2.07
16	4.49	3.63	3.24	3.01	2.85	2.74	2.66	2.59	2.54	2.49	2.42	2.35	2.28	2.23	2.19	2.15	2.11	2.06	2.01
17	4.45	3.59	3.20	2.96	2.81	2.70	2.61	2.55	2.49	2.45	2.38	2.31	2.23	2.18	2.15	2.10	2.06	2.01	1.96
18	4.41	3.55	3.16	2.93	2.77	2.66	2.58	2.51	2.46	2.41	2.34	2.27	2.19	2.14	2.11	2.06	2.02	1.97	1.92
19	4.38	3.52	3.13	2.90	2.74	2.63	2.54	2.48	2.42	2.38	2.31	2.23	2.16	2.11	2.07	2.03	1.98	1.93	1.88
20	4.35	3.49	3.10	2.87	2.71	2.60	2.51	2.45	2.39	2.35	2.28	2.20	2.12	2.07	2.04	1.99	1.95	1.90	1.84
21	4.32	3.47	3.07	2.84	2.68	2.57	2.49	2.42	2.37	2.32	2.25	2.18	2.10	2.05	2.01	1.96	1.92	1.87	1.81
22	4.30	3.44	3.05	2.82	2.66	2.55	2.46	2.40	2.34	2.30	2.23	2.15	2.07	2.02	1.98	1.94	1.89	1.84	1.78
23	4.28	3.42	3.03	2.80	2.64	2.53	2.44	2.37	2.32	2.27	2.20	2.13	2.05	2.00	1.96	1.91	1.86	1.81	1.76
24	4.26	3.40	3.01	2.78	2.62	2.51	2.42	2.36	2.30	2.25	2.18	2.11	2.03	1.97	1.94	1.89	1.84	1.79	1.73
25	4.24	3.39	2.99	2.76	2.60	2.49	2.40	2.34	2.28	2.24	2.16	2.09	2.01	1.96	1.92	1.87	1.82	1.77	1.71
30	4.17	3.32	2.92	2.69	2.53	2.42	2.33	2.27	2.21	2.16	2.09	2.01	1.93	1.88	1.84	1.79	1.74	1.68	1.62
40	4.08	3.23	2.84	2.61	2.45	2.34	2.25	2.18	2.12	2.08	2.00	1.92	1.84	1.78	1.74	1.69	1.64	1.58	1.51
60	4.00	3.15	2.76	2.53	2.37	2.25	2.17	2.10	2.04	1.99	1.92	1.84	1.75	1.69	1.65	1.59	1.53	1.47	1.39
120	3.92	3.07	2.68	2.45	2.29	2.18	2.09	2.02	1.96	1.91	1.83	1.75	1.66	1.60	1.55	1.50	1.43	1.35	1.25
∞	3.84	3.00	2.60	2.37	2.21	2.10	2.01	1.94	1.88	1.83	1.75	1.67	1.57	1.51	1.46	1.39	1.32	1.22	1.00

Values of $F_{0.01}$ F-table



ν_2 = Degrees of Freedom for Denominator	ν_1 = Degrees of Freedom for Numerator																		
	1	2	3	4	5	6	7	8	9	10	12	15	20	25	30	40	60	120	∞
1	4,052	5,000	5,403	5,625	5,764	5,859	5,928	5,982	6,023	6,056	6,106	6,157	6,209	6,240	6,261	6,287	6,313	6,339	6,366
2	98.50	99.00	99.17	99.25	99.30	99.33	99.36	99.37	99.39	99.40	99.42	99.43	99.45	99.46	99.57	99.47	99.48	99.49	99.50
3	34.12	30.82	29.46	28.71	28.24	27.91	27.67	27.49	27.35	27.23	27.05	26.87	26.69	26.58	26.50	26.41	26.32	26.22	26.13
4	21.20	18.00	16.69	15.98	15.52	15.21	14.98	14.80	14.66	14.55	14.37	14.20	14.02	13.91	13.84	13.75	13.65	13.56	13.46
5	16.26	13.27	12.06	11.39	10.97	10.67	10.46	10.29	10.16	10.05	9.89	9.72	9.55	9.45	9.38	9.29	9.20	9.11	9.02
6	13.75	10.92	9.78	9.15	8.75	8.47	8.26	8.10	7.98	7.87	7.72	7.56	7.40	7.30	7.23	7.14	7.06	6.97	6.88
7	12.25	9.55	8.45	7.85	7.46	7.19	6.99	6.84	6.72	6.62	6.47	6.31	6.16	6.06	5.99	5.91	5.82	5.74	5.65
8	11.26	8.65	7.59	7.01	6.63	6.37	6.18	6.03	5.91	5.81	5.67	5.52	5.36	5.26	5.20	5.12	5.03	4.95	4.86
9	10.56	8.02	6.99	6.42	6.06	5.80	5.61	5.47	5.35	5.26	5.11	4.96	4.81	4.71	4.65	4.57	4.48	4.40	4.31
10	10.04	7.56	6.55	5.99	5.64	5.39	5.20	5.06	4.94	4.85	4.71	4.56	4.41	4.31	4.25	4.17	4.08	4.00	3.91
11	9.65	7.21	6.22	5.67	5.32	5.07	4.89	4.74	4.63	4.54	4.40	4.25	4.10	4.01	3.94	3.86	3.78	3.69	3.60
12	9.33	6.93	5.95	5.41	5.06	4.82	4.64	4.50	4.39	4.30	4.16	4.01	3.86	3.76	3.70	3.62	3.54	3.45	3.36
13	9.07	6.70	5.74	5.21	4.86	4.62	4.44	4.30	4.19	4.10	3.96	3.82	3.66	3.57	3.51	3.43	3.34	3.25	3.17
14	8.86	6.51	5.56	5.04	4.69	4.46	4.28	4.14	4.03	3.94	3.80	3.66	3.51	3.41	3.35	3.27	3.18	3.09	3.00
15	8.68	6.36	5.42	4.89	4.56	4.32	4.14	4.00	3.89	3.80	3.67	3.52	3.37	3.28	3.21	3.13	3.05	2.96	2.87
16	8.53	6.23	5.29	4.77	4.44	4.20	4.03	3.89	3.78	3.69	3.55	3.41	3.26	3.16	3.10	3.02	2.93	2.84	2.75
17	8.40	6.11	5.18	4.67	4.34	4.10	3.93	3.79	3.68	3.59	3.46	3.31	3.16	3.07	3.00	2.92	2.83	2.75	2.65
18	8.29	6.01	5.09	4.58	4.25	4.01	3.84	3.71	3.60	3.51	3.37	3.23	3.08	2.98	2.92	2.84	2.75	2.66	2.57
19	8.18	5.93	5.01	4.50	4.17	3.94	3.77	3.63	3.52	3.43	3.30	3.15	3.00	2.91	2.84	2.76	2.67	2.58	2.49
20	8.10	5.85	4.94	4.43	4.10	3.87	3.70	3.56	3.46	3.37	3.23	3.09	2.94	2.84	2.78	2.69	2.61	2.52	2.42
21	8.02	5.78	4.87	4.37	4.04	3.81	3.64	3.51	3.40	3.31	3.17	3.03	2.88	2.79	2.72	2.64	2.55	2.46	2.36
22	7.95	5.72	4.82	4.31	3.99	3.76	3.59	3.45	3.35	3.26	3.12	2.98	2.83	2.73	2.67	2.58	2.50	2.40	2.31
23	7.88	5.66	4.76	4.26	3.94	3.71	3.54	3.41	3.30	3.21	3.07	2.93	2.78	2.69	2.62	2.54	2.45	2.35	2.26
24	7.82	5.61	4.72	4.22	3.90	3.67	3.50	3.36	3.26	3.17	3.03	2.89	2.74	2.64	2.58	2.49	2.40	2.31	2.21
25	7.77	5.57	4.68	4.18	3.85	3.63	3.46	3.32	3.22	3.13	2.99	2.85	2.70	2.60	2.54	2.45	2.36	2.27	2.17
30	7.56	5.39	4.51	4.02	3.70	3.47	3.30	3.17	3.07	2.98	2.84	2.70	2.55	2.45	2.39	2.30	2.21	2.11	2.01
40	7.31	5.18	4.31	3.83	3.51	3.29	3.12	2.99	2.89	2.80	2.66	2.52	2.37	2.27	2.20	2.11	2.02	1.92	1.80
60	7.08	4.98	4.13	3.65	3.34	3.12	2.95	2.82	2.72	2.63	2.50	2.35	2.20	2.10	2.03	1.94	1.84	1.73	1.60
120	6.85	4.79	3.95	3.48	3.17	2.96	2.79	2.66	2.56	2.47	2.34	2.19	2.03	1.93	1.86	1.76	1.66	1.53	1.38
∞	6.63	4.61	3.78	3.32	3.02	2.80	2.64	2.51	2.41	2.32	2.18	2.04	1.88	1.77	1.70	1.59	1.47	1.32	1.00

t-table



v	$\alpha = 0.10$	$\alpha = 0.05$	$\alpha = 0.025$	$\alpha = 0.01$	$\alpha = 0.00833$	$\alpha = 0.00625$	$\alpha = 0.005$	v
1	3.078	6.314	12.706	31.821	38.204	50.923	63.657	1
2	1.886	2.920	4.303	6.965	7.650	8.860	9.925	2
3	1.638	2.353	3.182	4.541	4.857	5.392	5.841	3
4	1.533	2.132	2.776	3.747	3.961	4.315	4.604	4
5	1.476	2.015	2.571	3.365	3.534	3.810	4.032	5
6	1.440	1.943	2.447	3.143	3.288	3.521	3.707	6
7	1.415	1.895	2.365	2.998	3.128	3.335	3.499	7
8	1.397	1.860	2.306	2.896	3.016	3.206	3.355	8
9	1.383	1.833	2.262	2.821	2.934	3.111	3.250	9
10	1.372	1.812	2.228	2.764	2.870	3.038	3.169	10
11	1.363	1.796	2.201	2.718	2.820	2.981	3.106	11
12	1.356	1.782	2.179	2.681	2.780	2.934	3.055	12
13	1.350	1.771	2.160	2.650	2.746	2.896	3.012	13
14	1.345	1.761	2.145	2.624	2.718	2.864	2.977	14
15	1.341	1.753	2.131	2.602	2.694	2.837	2.947	15
16	1.337	1.746	2.120	2.583	2.673	2.813	2.921	16
17	1.333	1.740	2.110	2.567	2.655	2.793	2.898	17
18	1.330	1.734	2.101	2.552	2.639	2.775	2.878	18
19	1.328	1.729	2.093	2.539	2.625	2.759	2.861	19
20	1.325	1.725	2.086	2.528	2.613	2.744	2.845	20
21	1.323	1.721	2.080	2.518	2.602	2.732	2.831	21
22	1.321	1.717	2.074	2.508	2.591	2.720	2.819	22
23	1.319	1.714	2.069	2.500	2.582	2.710	2.807	23
24	1.318	1.711	2.064	2.492	2.574	2.700	2.797	24
25	1.316	1.708	2.060	2.485	2.566	2.692	2.787	25
26	1.315	1.706	2.056	2.479	2.559	2.684	2.779	26
27	1.314	1.703	2.052	2.473	2.553	2.676	2.771	27
28	1.313	1.701	2.048	2.467	2.547	2.669	2.763	28
29	1.311	1.699	2.045	2.462	2.541	2.663	2.756	29
inf.	1.282	1.645	1.960	2.326	2.394	2.498	2.576	inf.

STANDARD NORMAL DISTRIBUTION: Table Values Represent AREA to the LEFT of the Z score.

Z	.00	.01	.02	.03	.04	.05	.06	.07	.08	.09
0.0	.50000	.50399	.50798	.51197	.51595	.51994	.52392	.52790	.53188	.53586
0.1	.53983	.54380	.54776	.55172	.55567	.55962	.56356	.56749	.57142	.57535
0.2	.57926	.58317	.58706	.59095	.59483	.59871	.60257	.60642	.61026	.61409
0.3	.61791	.62172	.62552	.62930	.63307	.63683	.64058	.64431	.64803	.65173
0.4	.65542	.65910	.66276	.66640	.67003	.67364	.67724	.68082	.68439	.68793
0.5	.69146	.69497	.69847	.70194	.70540	.70884	.71226	.71566	.71904	.72240
0.6	.72575	.72907	.73237	.73565	.73891	.74215	.74537	.74857	.75175	.75490
0.7	.75804	.76115	.76424	.76730	.77035	.77337	.77637	.77935	.78230	.78524
0.8	.78814	.79103	.79389	.79673	.79955	.80234	.80511	.80785	.81057	.81327
0.9	.81594	.81859	.82121	.82381	.82639	.82894	.83147	.83398	.83646	.83891
1.0	.84134	.84375	.84614	.84849	.85083	.85314	.85543	.85769	.85993	.86214
1.1	.86433	.86650	.86864	.87076	.87286	.87493	.87698	.87900	.88100	.88298
1.2	.88493	.88686	.88877	.89065	.89251	.89435	.89617	.89796	.89973	.90147
1.3	.90320	.90490	.90658	.90824	.90988	.91149	.91309	.91466	.91621	.91774
1.4	.91924	.92073	.92220	.92364	.92507	.92647	.92785	.92922	.93056	.93189
1.5	.93319	.93448	.93574	.93699	.93822	.93943	.94062	.94179	.94295	.94408
1.6	.94520	.94630	.94738	.94845	.94950	.95053	.95154	.95254	.95352	.95449
1.7	.95543	.95637	.95728	.95818	.95907	.95994	.96080	.96164	.96246	.96327
1.8	.96407	.96485	.96562	.96638	.96712	.96784	.96856	.96926	.96995	.97062
1.9	.97128	.97193	.97257	.97320	.97381	.97441	.97500	.97558	.97615	.97670
2.0	.97725	.97778	.97831	.97882	.97932	.97982	.98030	.98077	.98124	.98169
2.1	.98214	.98257	.98300	.98341	.98382	.98422	.98461	.98500	.98537	.98574
2.2	.98610	.98645	.98679	.98713	.98745	.98778	.98809	.98840	.98870	.98899
2.3	.98928	.98956	.98983	.99010	.99036	.99061	.99086	.99111	.99134	.99158
2.4	.99180	.99202	.99224	.99245	.99266	.99286	.99305	.99324	.99343	.99361

Practice Problems

Q1. If two fair dice are rolled, what is the probability that sum of the numbers appearing is 9, given that at least one 4 appears?

Q2. Suppose X has the following probability mass function $p(0) = 0.2$, $p(1) = 0.5$, $p(2) = 0.3$. Calculate $E[X^2]$.

Q3. A fair die is rolled. If even number appears then a coin is flipped. Draw the probability tree and find the probability of getting -

(i) An even number & Tail.
(ii) 6 and Head
(iii) 3 and Head.

Q4. If the probability is 0.05 that a certain wide-flange column will fail under a given axial load, what are the probabilities that among 16 such columns

(a) At most two will fail
(b) At least four will fail

Q5. Compute Mean and Variance of X, when X represents the outcome when we roll a fair die.

Q6. Average income of a group of employees in a company is Rs.12,000 with standard deviation of Rs. 1,100. Find the percentage of employees earning –

i) Between Rs. 9,000 and 11,000
ii) More than Rs. 13,000
iii) Less than Rs. 10,000

Q7. Average IQ of a class is 120 with a variance 64. Find -

i) Minimum score of the 3% students who have a higher IQ.
ii) Maximum score of the 5% students who have a lower IQ.

Q8. In a town on an average 10 accidents occur within a span of 50 days. Assuming the no. of accidents per day follow Poisson distribution, find the probability that there will be 3 or more accidents in a day. (Note : e = 2.7183)

Q9. Using normal probabilities to determine the mean fill of jars -
The actual amount of instant coffee that a filling machine puts into “4-ounce” jars may be looked upon as a random variable having a normal distribution with σ = 0.04 ounce. If only 2 % of the jars are to contain less than 4 ounces, what should be the mean fill of these jars ?

Q10. Suppose the probability that an item produced by a certain machine will be defective is 0.1. Find the probability that a sample of 10 items will contain at most one defective item. Assume that the quality of successive items is independent.

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

BSDCHZC355
Statistical Inference & Applications

Shaibal Kr. Sen

Session 02

innovate achieve lead

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

STATISTICAL INFERENCES

&

APPLICATIONS

Scheduled Topics to be covered

Review of Testing of Hypothesis

Parametric Tests –

(i) Test of mean for single population using Z-test and t-test

(ii) Testing the difference between two means using Z-test

(iii) Testing the difference between two means using t-test

(iv) Testing of hypothesis for proportion of two populations

Concept of Non-Parametric Tests

Supposed or Proposed explanation made on the basis of limited evidence is a starting point for further investigation. In the context of Probability, Hypothesis is a statement or claim about the population parameter. Correctness of which is subject to test prior to accepting / rejecting – Accept if the test report of the claim is favourable and reject if the test report of the claim is unfavourable. Explained with relevant examples during the session.

Null and Alternate Hypothesis

If a certain claim is made about a population parameter then it is compared against another hypothesis, which is accepted when original hypothesis is rejected.

Generally the hypothesis which we wish to establish is taken as alternate hypothesis and denoted by H_1. The hypothesis to be contested is called null hypothesis, denoted by H_0.

Type I error or α error **:** There is a chance that null hypothesis is rejected when it is true, that is, we have committed type I error.

Probability of Type I error is $\alpha = P(H_0 \text{ is rejected} \mid H_0 \text{ is true})$. This is also called level of significance.

Type II error or β error **:** There is a chance that null hypothesis is accepted when it is false, that is, we have committed type II error.

Probability of Type II error is $\beta = P(H_0 \text{ is accepted} \mid H_0 \text{ is false})$.

Types of Tests

A statistical rule which decides whether to accept or reject the null hypothesis on the basis of data. Following Tests are commonly employed –

Parametric Tests **:** Based on the assumption of some probability distribution.

Non-Parametric Tests **:** Not based on any assumption of probability distribution. It is assumed that the data do not follow any probability distribution which is not characterized by any parameters - Distribution - free tests – Chi-Square Test.

Parametric tests

It is assumed that the data do follow some probability distribution which is characterized by any parameters.

Large Sample Test → $n \geq 30$

Standard Normal Test → Z-Test

Small Sample Test → $n < 30$

Student's t-test → Unpaired t-Test, Paired t-Test

Parametric tests

Z-test

This is a test based on Standard Normal Distribution

Used for testing the

- Mean of a single population (μ)
- Difference between means of two populations ($\mu_1 - \mu_2$)
- Proportion of a single population (P)
- Difference between proportions of two populations ($P_1 - P_2$)

One sided and both sided intervals

One Sided

1a) If $H_o : \mu \leq \mu_0$

$H_1 : \mu > \mu_0$ (ie. it is greater than μ_0)

Lower tail is Acceptance region & Upper tail is Rejection region

1b) If $H_o : \mu \geq \mu_0$

$H_1 : \mu < \mu_0$ (ie. it is lesser than μ_0)

Upper tail is Acceptance region & Lower tail is Rejection region

Both Sided

2) If $H_o : \mu = \mu_0$

$H_1 : \mu \neq \mu_0$ (ie. it is either greater or lesser than μ_0)

Central Region (CR) is Acceptance Region and both ends beyond CR is Rejection Region

Test of Mean for single population

Ex 1 : It is claimed that sports-car owners drive on the average 18,580 kms per year. A consumer firm believes that the average mileage is not same as the claim. To check, the consumer firm obtained information from randomly selected 10 sports-car owners that resulted in a sample mean of 17,352 kms with a sample standard deviation of 2,012 kms. What can be concluded about this claim at 5% level of significance (LOS)?

Soln : Given $\mu_0 = 18{,}580$ kms, $s = 2{,}012$ kms, and $n = 10$, $x = 17{,}352$ kms

Here, $H_0 : \mu = \mu_0 = 18{,}580$ and $H_1 : \mu \neq \mu_0 \neq 18{,}580$

It is a parametric test, $t = \sqrt{n}\,(x - \mu_0) / (s) = (17{,}352 - 18{,}580) / (2012/\sqrt{10})$

$$= (-1228) / 636.25 = -1.93$$

Critical value for $\alpha = 0.05$ and for two tailed test for $\alpha/2 = 0.025$ is 2.262 for $n - 1 = 9$ degree of freedom (obtained from t-table)

Since $| t | = 1.929 < 2.262$, Hypothesis is accepted ie. sports car owner's claim is accepted.

Ex 2 : Average Height of 10 males of a given locality is found to be 66 inches & variance is 10 in^2. Is it reasonable to believe that the average height is greater than 64 inches? Test at 5% level of significance. (Obtained from t table, t at 9 d.f. at 5% level of significance is 1.833).

Soln : Given; $n = 10$, $\mu_0 = 64$, $\alpha = 0.05$, $\bar{x} = 66$, $s = \sqrt{10}$, $t_{0.05,\,9} = 1.833$

Null hypothesis should be $H_0 : \mu \geq 64$ and alternative hypothesis $H_1 : \mu < 64$

Test statistic $t = \{(\bar{x} - \mu_0)\sqrt{n}\} / s = (66 - 64)\sqrt{10} / \sqrt{10} = 2$

Now, as the critical value of $|t_{0.05,9}|$ is 1.833, the acceptance region is $t = -1.833$ to ∞ and rejection region is $-\infty$ to -1.833 as shown in the diagram (in next slide). As the test statistic (2) is within the acceptance region, the hypothesis that average height is greater than 64 inches is accepted.

Ex 3 : A sample of 40 sales receipts from a grocery store has average \$137 and σ = \$30.2. Use these values to test whether or not the mean of sales at the grocery store are different from \$150.

Soln : Step 1 : Set the null and alternative hypothesis

$$H_0 : \mu = 150$$

$$H_1 : \mu \neq 150$$

Step 2 : Calculate the test statistic

$$Z = \sqrt{n}\,(\bar{x} - \mu_0) / \sigma = (137 - 150) / 30.2 / \sqrt{40} = -2.722$$

Step 3 : Set Rejection Region

Looking at the the picture below, we need to put half of α in the left tail and the other half of α in the right tail . Thus R : $|Z| > 2.575$ (assumed $\alpha = 1\%$)

Make the normal graph showing both regions beyond 2.575 (0.005) – as shown above

Step 4 : Conclude

We can see that $|-2.722| = 2.722 > 2.575$, thus our test statistic is in the rejection region. Therefore we reject the null hypothesis in favor of the alternative. We can conclude that the mean is significantly different from \$150, thus it is proven that the mean sales at the grocery store is not \$150.

Ex 4 : A sample of 35 sales receipts of a grocery store reveals average sale of Rs. 10000 at σ = Rs. 300. Use these values to test whether or not the mean sale at the grocery store is different from Rs. 11000. Take level of significance (LOS) as 1%. [Given $Z_{critical} = 2.575$]

Soln : Step 1 **:** Set the null and alternative hypothesis

$$H_0 : \mu = 11000$$

$$H_1 : \mu \neq 11000$$

Step 2 **:** Calculate the test statistic

$$Z = \sqrt{n}\ (\underline{x} - \mu_0) / \sigma = (10000 - 11000) / 300 / \sqrt{35} = - 1000 / 50.71 = 19.73$$

Step 3 **:** Set Rejection Region

Looking at the the picture below, we need to put half of α in the left tail and the other half of α in the right tail . Thus R **:** $| Z | > 2.575$ (assumed $\alpha = 1\%$)

Make the normal graph showing both regions beyond 2.575 (0.005) – as shown above

Step 4 **:** Conclude

We can see that $|- 2.722 | = 2.722 > 2.575$, thus our test statistic is in the rejection region. Therefore we reject the null hypothesis in favor of the alternative. We can conclude that the mean is significantly different from $150, thus it is proven that the mean sales at the grocery store is not $150.

Ex 5 **:** A company mfrs steel bars. If the production process is working properly, it turns out steel bars with an average length of at least 2.8 mts with a std deviation of 0.2 mts (as determined from Engg specs of the prodn eqpt involved). Longer steel bars can be used or altered, but the shorter bars must be scrapped. A sample of 25 bars is selected from the prodn line. The sample indicates an average length of 2.73 mts. The company wishes to test the hypothesis at the 0.05 level of significance, what decision would it make using the critical value approach to hypothesis testing ?

Soln **:** Step 1 **:** Set the null and alternative hypothesis

$$H_0 : \mu \geq 2.8$$

$$H_1 : \mu < 2.8$$

Step 2 **:** Calculate the test statistic

$$t = \sqrt{n}\,(x - \mu\ ) / \sigma = \sqrt{25}\,(2.73 – 2.8) / 0.2$$

$$= -1.75$$

Step 3 **:** Set Rejection Region - At $\alpha = .05$ reject H_0 if $t_{0.05,24}$ (-1.711) > -1.75

Step 4 **:** Conclude - We can see that -1.75 < -1.711, thus our test statistic is in the rejection region (at = 0.05). Therefore, we reject the null hypothesis. We can conclude that average length of steel bars are less than 2.8 mts.

Ex 6 : A car manufacturing company claims that their cars do not require servicing before running for on an average 10,000 miles. To verify the claim, the manufacturer tests 20 cars under simulated conditions and gets a mean breakdown of 9500 miles with a standard deviation of 1150 miles. What conclusion can be drawn at a significance level of 5%.

Soln : Given : $\overline{x} = 9500$, $\mu = 10000$, $n = 20$, $s = 1150$

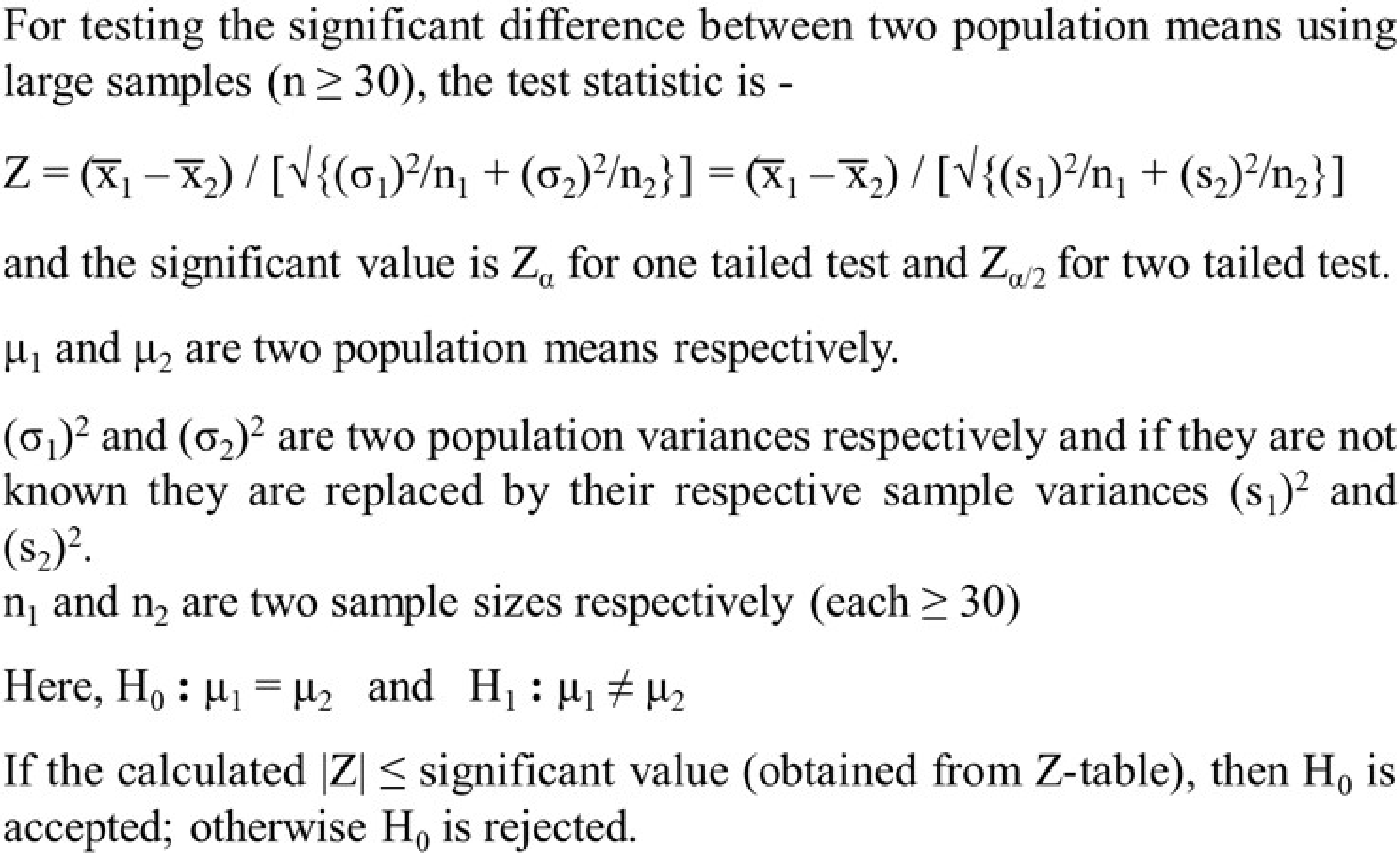
$H_0 : \mu \geq \mu$ and $H1 : \mu < \mu$ Test statistic $t = \sqrt{n}\,(\overline{x} - \mu) / s$

$= (9500 - 10000) / (1150/\sqrt{20}) = -500 / 257.15 = -1.94$

From t-table we find, $t_{0.05,\ df = 19} = -1.729$.

Since the test statistic value is beyond the acceptance region, the Hypothesis can not be accepted. In other words the manufacturer’s claim is rejected.

Testing the difference between two means using Z-test



For testing the significant difference between two population means using large samples ($n \geq 30$), the test statistic is -

$$Z = (\overline{x}_1 - \overline{x}_2) / [\sqrt{\{(\sigma_1)^2/n_1 + (\sigma_2)^2/n_2\}}] = (\overline{x}_1 - \overline{x}_2) / [\sqrt{\{(s_1)^2/n_1 + (s_2)^2/n_2\}}]$$

and the significant value is Z_α for one tailed test and $Z_{\alpha/2}$ for two tailed test.

μ_1 and μ_2 are two population means respectively.

$(\sigma_1)^2$ and $(\sigma_2)^2$ are two population variances respectively and if they are not known they are replaced by their respective sample variances $(s_1)^2$ and $(s_2)^2$.

n_1 and n_2 are two sample sizes respectively (each ≥ 30)

Here, $H_0 : \mu_1 = \mu_2$ and $H_1 : \mu_1 \neq \mu_2$

If the calculated $|Z| \leq$ significant value (obtained from Z-table), then H_0 is accepted; otherwise H_0 is rejected.

Ex 7 : In a survey of buying habits. 400 women shopper's are chosen at random in super Market A located in a certain city. Their average weekly food expenditure is Rs.250 with a SD of Rs.40. For 400 women shopper's are chosen at random in super Market B in another city. The average weekly food expenditure is Rs.220 with a SD of Rs.55.

Test at 1% α ie. Level of Significance (LOS) whether the average weekly food expenditure of the two populations of shoppers are equal.

Soln : Given $n_1 = 400, n_2 = 400, \overline{x}_1 = 250, \overline{x}_2 = 220, s_1 = 40, s_2 = 55$

LOS $\alpha = 0.01$, corresponding value from Z-table = 2.58

Now, Null Hypo $H_0 : \mu_1 = \mu_2$ and Alternate Hypo $H_1 : \mu_1 \neq \mu_2$ (2-tailed test)

Test stat $Z = (\overline{x}_1 - \overline{x}_2) / \{\sqrt{(s_1^2 / n_1 + s_2^2 / n_2)}\} = (250 - 220) / \{\sqrt{(40^2 / 400 + 55^2 / 400)}\}$

$= 30 / \sqrt{11.5625} = 30 / 3.4 = 8.82$

As, Z_{stat} (8.82) > Z_{tab} (2.58), Null Hypothesis is rejected (ie. Food expenditure of 2 population of shoppers are not equal).

Ex 8 **:** There is an assumption that there is no significant difference between boys and girls wrt intelligence. Tests are conducted on two groups and the following are the observations.

	Size	Mean	SD
Girls	60	75	8
Boys	100	73	10

Validate this at 5% Level of Significance.

Soln **:** Ho : $\mu_1 = \mu_2$ and $\mu_1 \neq \mu_2$ (Two tailed test)

Test statistic $Z = \{(\bar{x}_1 - \bar{x}_2)\} / [\sqrt{\{s_1^2/n_1 + s_2^2/n_2\}}]$

$= (75 - 73) / [\sqrt{(8^2/60 + 10^2/100)}] = 1.3912$

Since test statistic value 1.3912 falls within acceptance region the hypothesis is accepted.

Ex **9 :** The researchers reported the following sample statistics.
In a sample of 45 women dining with other women, the average number of calories ordered was 850, and the standard deviation was 252.
In a sample of 32 women dining with men, the average number of calories ordered was 719, and the standard deviation was 322.
(i) At 5% LOS can we conclude that average calories ordered by both the group of women are same?
(ii) Also evaluate the P-value.

Soln : (i) Using the steps applied in previous numerical we proceed to find

Test statistic, $Z_{stat} = \{(x_1 - x_2)\} / [\sqrt{\{(s_1)^2/n_1 + (s_2)^2/n_2\}}]$

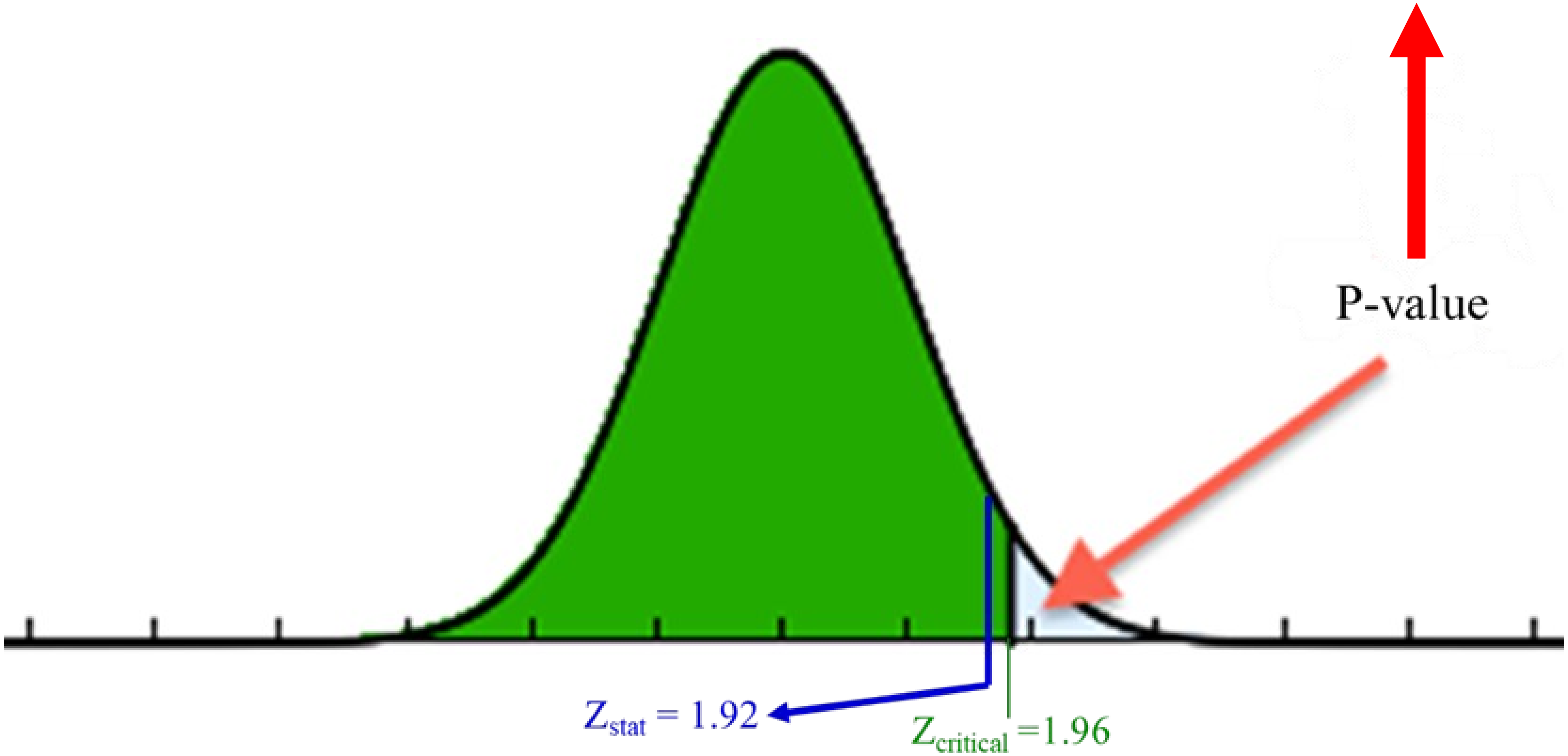
$= (850 - 719) / [\sqrt{\{(252)^2 / 45 + (322)^2 / 32\}}] \approx 131 / 72.47 \approx 1.92$

$Z_{critical}$ (obtained from Z-table) = 1.96

Since, Z_{stat} (1.92) < $Z_{critical}$ (1.96), null hypothesis is accepted.

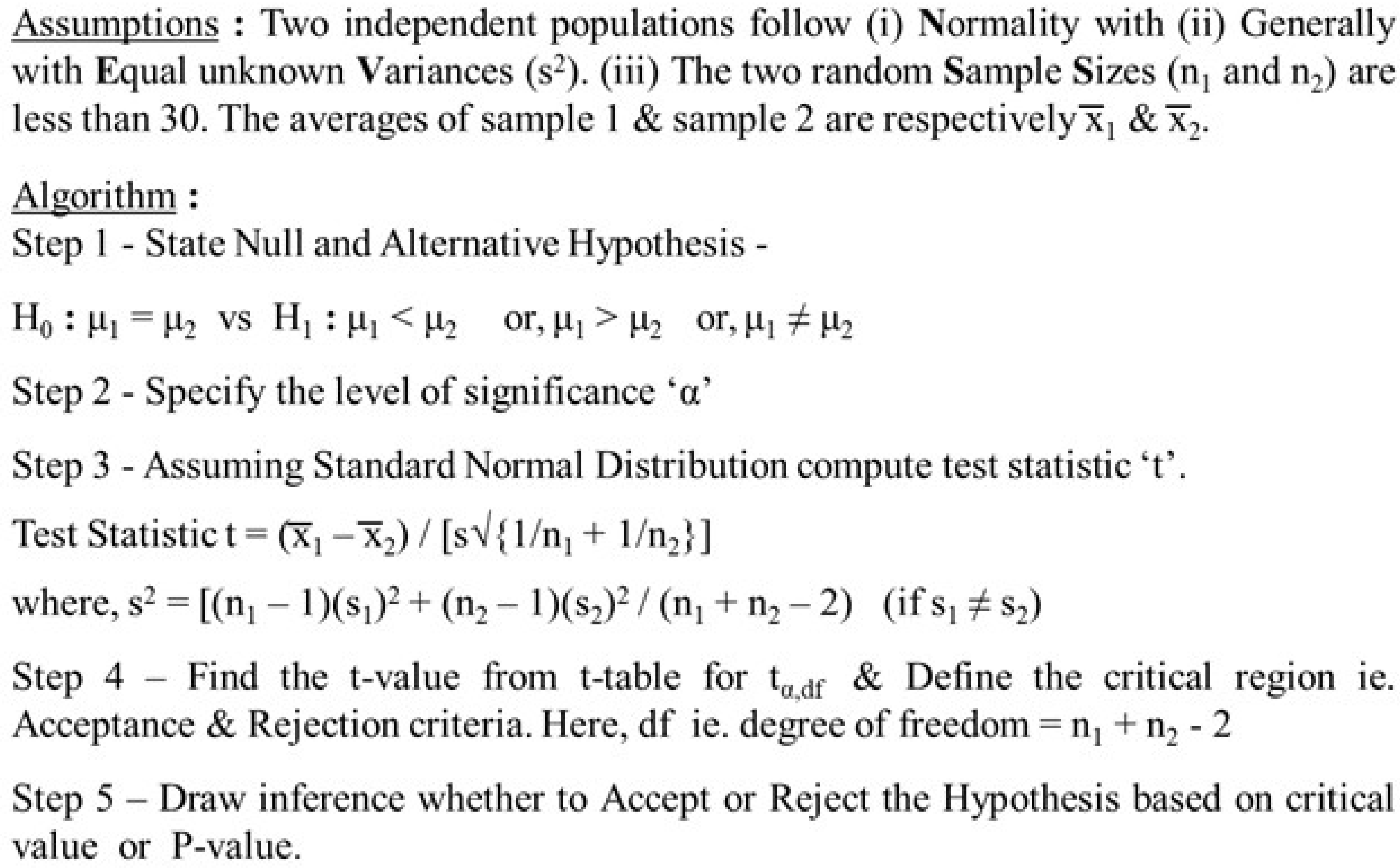
(ii) Area to the left of Zstat (1.92) = 0.9726. Hence area to the right of 1.92 is -

1 – 0.9726 = 0.0274



P-value = 0.0274 which is more than the significance level 0.025. Hence, we accept null hypothesis and reject alternative hypothesis.

Testing the difference between two means using t-test



Assumptions : Two independent populations follow (i) **N**ormality with (ii) Generally with **E**qual unknown **V**ariances (s^2). (iii) The two random **S**ample **S**izes (n_1 and n_2) are less than 30. The averages of sample 1 & sample 2 are respectively $\overline{x}_1$ & $\overline{x}_2$.

Algorithm **:**

Step 1 - State Null and Alternative Hypothesis -

$H_0 : \mu_1 = \mu_2$ vs $H_1 : \mu_1 < \mu_2$ or, $\mu_1 > \mu_2$ or, $\mu_1 \neq \mu_2$

Step 2 - Specify the level of significance ‘α’

Step 3 - Assuming Standard Normal Distribution compute test statistic ‘t’.

Test Statistic $t = (\overline{x}_1 - \overline{x}_2) / [s\sqrt{\{1/n_1 + 1/n_2\}}]$

where, $s^2 = [(n_1 - 1)(s_1)^2 + (n_2 - 1)(s_2)^2 / (n_1 + n_2 - 2)$ (if $s_1 \neq s_2$)

Step 4 – Find the t-value from t-table for $t_{\alpha,df}$ & Define the critical region ie. Acceptance & Rejection criteria. Here, df ie. degree of freedom = $n_1 + n_2 - 2$

Step 5 – Draw inference whether to Accept or Reject the Hypothesis based on critical value or P-value.

Ex 10 : The manager of a courier service believes that packets delivered at the beginning of the month are heavier than those delivered at the end of month. As an experiment, he weighed a random sample of 15 packets at the beginning of the month and found that the mean weight was 5.25 kg. A randomly selected 10 packets at the end of the month had a mean weight of 4.26 kg. It was observed from the past experience that the sample variances are 1.20 kg^2 and 1.15 kg^2. Determine - At 5% level of significance, can it be concluded that the packets delivered at the beginning of the month weigh more?

Soln **:** Given : $\overline{x}_1 = 5.25$, $\overline{x}_2 = 4.26$, $n_1 = 15$, $n_2 = 10$, $(s_1)^2 = 1.2$, $(s_2)^2 = 1.15$

Degree of freedom = $n_1 + n_2 - 2 = 15 + 10 - 2 = 23$;

from t-table $t_{0.05/2,\ 23} = 2.069$ (because it is both sided test)

$s = \sqrt{[(n_1 - 1)(s_1)^2 + (n_2 - 1)(s_2)^2 / (n_1 + n_2 - 2)]} = \sqrt{[(14 \times 1.2 + 9 \times 1.15) / 23]}$

$= 1.086$

Now, Null Hypothesis $H_0 : \mu_1 = \mu_2$ and Alt Hypo $H_1 : \mu_1 \neq \mu_2$

Test statistic $t = (x_1 - x_2) / \{s\sqrt{(1/n_1 + 1/n_2)}\}$

$$= (5.25 - 4.26) / \{1.086\sqrt{(1/15 + 1/10)}\} = 2.233$$

Since, t_{stat} (2.233) > $t_{critical}$ (2.069) H_0 is rejected.

Ex 11 **:** As a manager of finance you are assigned the task of choosing the best product in terms of life of the product. Use 5% level of significance.

Product A - Mean 1456 hours with SD 423, size 10

Product B - Mean 1280 hours with SD 398, size 17

Soln **:** $H_0 : \mu_1 = \mu_2$ and $H_1 : \mu_1 \neq \mu_2$

$s^2 = (n_1 - 1)s_1^2 + (n_2 - 1)s_2^2 / (n_1 + n_2 - 2) = 9(423)^2 + 16(398)^2 / (10 + 17 - 2)$

$= 1,65,793$; Hence, $s = 407.18$

Test stat $t = (\bar{x}_1 - \bar{x}_2) / \{s\sqrt{(1/n_1 + 1/n_2)}\}$

$= \{(1456 - 1280)\} / \{407.18(1/10 + 1/17)\} = 1.085$

Degree of Freedom $= n_1 + n_2 - 2 = 10 + 17 - 2 = 25$

We obtain from t-table, $t_{0.05/2,\ 25} = 2.06$

Since test statistic t falls within acceptance region, we accept the null hypothesis.

Ex 12 : Random samples of 15 and 10 were selected from two thermocouples. The sample means were 315, 303 and sample standard deviations were 3.8, 4.9 respectively. Test whether there is any significant difference in the means of two thermocouples at 5% level of significance

Soln : Given $\overline{x}_1 = 315$, $\overline{x}_2 = 303$, $s_1 = 3.8$, $s_2 = 4.9$, $n_1 = 15$, $n_2 = 10$

LOS $\alpha = 0.05$, Degree of freedom $= n_1 + n_2 – 2 = 15 + 10 – 2 = 23$

$s^2 = (n_1 – 1)(s_1)^2 + (n_2 – 1)(s_2)^2 / (n_1 + n_2 – 2)$

$= 14(3.8)^2 + 9(4.9)^2 / (15 + 10 – 2) = 18.185$; Hence, $s = 4.26$

$t_{0.05/2,\ 23} = 2.069$ (From t-table)

$H_0 : \mu_1 = \mu_2$ and $\underline{H}_1 : \underline{\mu}_1 \neq \mu_2$

Test statistic $t = (x_1 – x_2) / \{s\sqrt{(1/n_1 + 1/n_2)}\}$

$= (315 – 303) / \{4.26\sqrt{1/15 + 1/10}\}] = 12 / 1.739 = 6.9$

Test statistics value is beyond acceptance region, so we can not accept the null hypothesis.

Testing of Hypothesis for proportion of two populations

Ex 13 **:** Break up of no. of disconnections of BSNL telephones in the 2 cities – Bangalore & Delhi are respectively, 387 disconnected out of 1500 connections & 310 disconnected out of 1200 connections. Ascertain if there is significant difference between the two proportions of telephone disconnection by BSNL at a confidence level of 99% (LOS = 1%).

Soln **:** Given; $x_1 = 387$, $n_1 = 1500$, $x_2 = 310$, $n_2 = 1200$

So, $p_1 = x_1 / n_1 = 387 / 1500 = 0.258$ and $p_2 = x_2 / n_2 = 310 / 1200 = 0.2583$

$P = (x_1 + x_2) / (n_1 + n_2) = (387 + 310) / (1500 + 1200) = 697 / 2700 = 0.2581$

$Q = 1 - P = 1 – 0.2581 = 0.7419$

Now, H_0 **:** $p_1 = p_2$ and H_1: $p_1 \neq p_2$ (two tailed test)

$Z_{stat} = (p_1 – p_2) / [\sqrt{\{(PQ)(1 / n_1 + 1/n_2)\}}]$

$= (0.258 – 0.2583) / [\sqrt{\{(0.2581 x 0.7419)(1/1500 + 1/1200)\}}] = - 0.0003 / 0.0169$

$= - 0.017$

Z_{tab} at 1% LOS = 2.58. Since $Z_{test} < Z_{tab}$ H_0 is accepted ie. no significant difference in the proportion of telephone disconnection between Bangalore & Delhi.

Ex 14 **:** It is found that 290 errors in the randomly selected 400 lines of code from Team A and 160 errors in 300 lines of code from Team B. At 5% LOS can we say that Team B's performance is superior to that of Team A?

Soln : $p_1 = x_1 / n_1 = 290 / 400 = 0.725$, $p_2 = x_2 / n_2 = 160 / 300 = 0.533$

$P = (x_1 + x_2) / (n_1 + n_2) = (290 + 160) / (400 + 300) = 0.6428$; so $1 – P = 0.3572$

Null Hypo H_0 **:** $p_2 \geq p_1$ and Alternate Hypo H_1 **:** $p_2 < p_1$

$Z_{test} = (p_1 – p_2) / [\sqrt{}\{P(1 – P)(1/n_1 + 1/n_2)\}]$

$= (0.725 – 0.533) / [\sqrt{}\{(0.6428x0.3572)((1/400 + 1/300)\}] = 5.33$

At 5% LOS $Z_{tab} = -1.645$.

Since test stat is in the Acceptance region, we accept the hypothesis & infer that Team B's performance is superior to that of Team A.

Non - Parametric tests

It is assumed that the data do not follow any probability distribution which is not characterized by any parameters.

Chi - Square Test is Distribution - free tests

Testing of Hypothesis : Chi square Test
Non-Parametric Test

- Independence of Two categorical Variables
- Goodness-of-fit Used for Discrete Distribution

Should be applied only for frequencies. Not for Percentages, Ratios, Mean etc.

The Hypothesis to be tested for independence will be –

H_0 : The two categorical variables may be Independent (may not be associated).

H_1 : The two categorical variables may not be Independent (may be associated).

Statistical Test is $\chi^2 = \sum_{i=1}^{k} (O_i - E_i)^2 / E_i$, k = r x c no. of cells (total no. of cells in r x c contingency table)

The test-statistic follows Chi-square distribution with (r – 1)(c – 1) degrees of freedom where, r = no. of rows & c = no. of columns

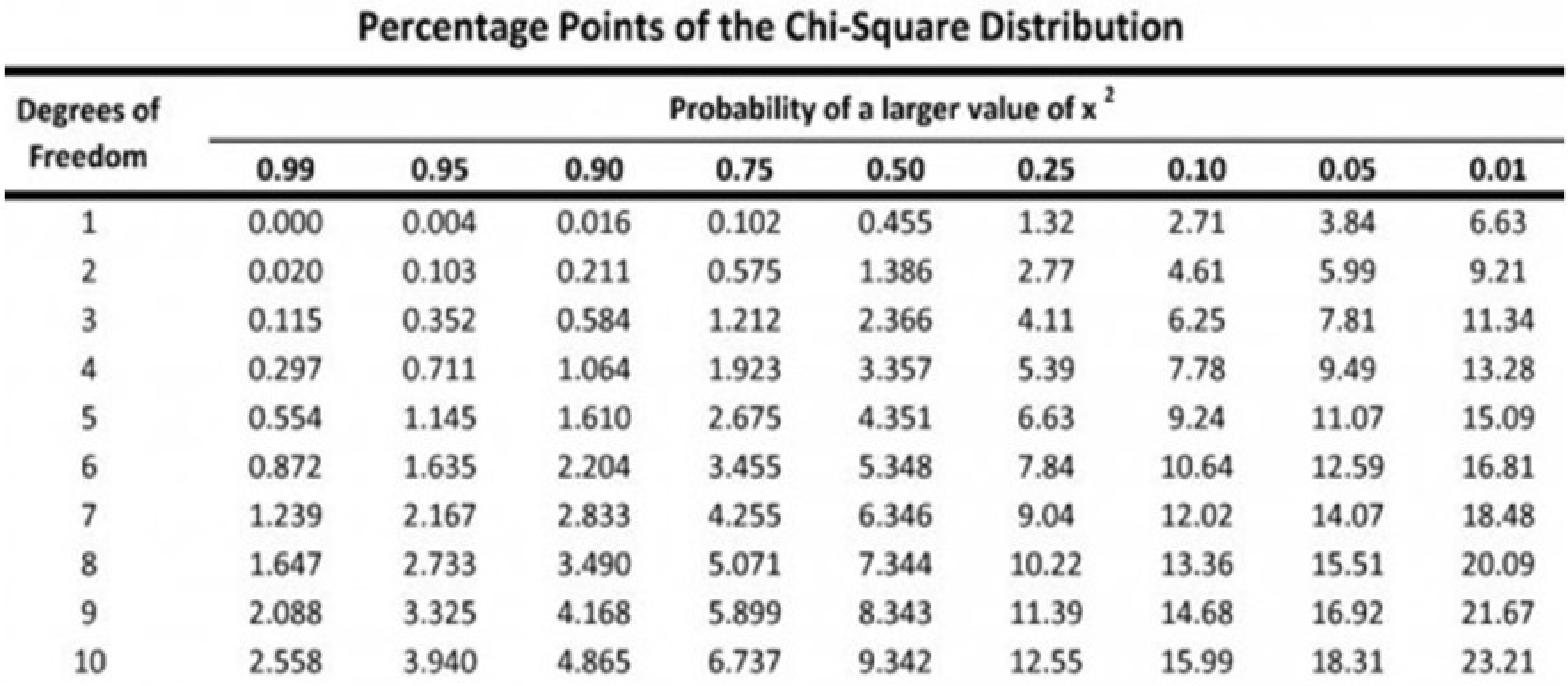
Chi-square is calculated by –

$$\chi^2 = \sum_{i=1}^{k}(O_i - E_i)^2 / E_i \approx \chi^2[(r-1)(c-1)]$$

where, k = r x c is total no. of cells in the r x c contingency table, r is total no. of rows and c is total no. of columns.

Expected frequencies $E_{ij} = r_i c_j / n$, for i = 1, 2,3,…., m & j = 1, 2, 3,…., n

r_i = ith row total, c_j = jth column total and n = overall total



Percentage Points of the Chi-Square Distribution

Degrees of Freedom	Probability of a larger value of x^2								
	0.99	0.95	0.90	0.75	0.50	0.25	0.10	0.05	0.01
1	0.000	0.004	0.016	0.102	0.455	1.32	2.71	3.84	6.63
2	0.020	0.103	0.211	0.575	1.386	2.77	4.61	5.99	9.21
3	0.115	0.352	0.584	1.212	2.366	4.11	6.25	7.81	11.34
4	0.297	0.711	1.064	1.923	3.357	5.39	7.78	9.49	13.28
5	0.554	1.145	1.610	2.675	4.351	6.63	9.24	11.07	15.09
6	0.872	1.635	2.204	3.455	5.348	7.84	10.64	12.59	16.81
7	1.239	2.167	2.833	4.255	6.346	9.04	12.02	14.07	18.48
8	1.647	2.733	3.490	5.071	7.344	10.22	13.36	15.51	20.09
9	2.088	3.325	4.168	5.899	8.343	11.39	14.68	16.92	21.67
10	2.558	3.940	4.865	6.737	9.342	12.55	15.99	18.31	23.21

Testing of Hypothesis - Chi-square test : Independence

Interpretation -

H_0 **:** Smoking habit and Cancer of lung may be independent (may not be associated).

H_1 **:** Smoking habit and Cancer of lung may not be independent (may be associated).

$\chi^2 = 5.555$

df = 1, Critical value at = 0.05 is 3.841, P = 0.035.

Inference **:** There may be association between smoking and Cancer of lung.

Ex 15 : Three pension plans independent of job classification. Use α = 0.05. The opinion of a random sample of 500 employees are shown below -

	Pension Plan			
Job Classification	1	2	3	Total
Salaried Workers	166	86	68	320
Hourly Workers	84	64	32	180
Total	250	150	100	500

$E_1 = \{(r_1c_1) / n\} = \{(320\text{x}250) / 500\} = 106.24$

$E_2 = \{(r_1c_2) / n\} = \{(320\text{x}150) / 500\} = 96.00$

$E_3 = \{(r_1c_3) / n\} = \{(320\text{x}100) / 500\} = 64.00$

$E_4 = \{(r_2c_1) / n\} = \{(180\text{x}250) / 500\} = 90.00$

$E_5 = \{(r_2c_2) / n\} = \{(180\text{x}150) / 500\} = 54.00$

$E_6 = \{(r_2c_3) / n\} = \{(180\text{x}100) / 500\} = 36.00$

Sl No.	O_i	E_i	$O_i - E_i$	$(O_i - E_i)^2$	$(O_i - E_i)^2 / E_i$
1	166	106.24	59.76	3571.26	33.62
2	86	96	-10	100	1.04
3	68	64	4	16	0.25
4	84	90	-6	36	0.40
5	64	54	10	100	1.85
6	32	36	-4	16	0.44
				Chi-square value	37.6

$\chi^2 = \sum_{i=1}^{k} (O_i - E_i)^2 / E_i$, k = r x c no. of cells (total no. of cells in r x c contingency table)

H_0 : Job satisfaction and pension plan may be independently distributed (not associated).

H_1 : Job satisfaction and pension plan may not be independently distributed (Associated).

$\chi^2 = 37.6$

df = (2 - 1)(3 – 1) = 2

$\chi^2_{0.05,\ df2}$ (from table) for LOS 0.05 = 5.99

Inference : Reject H_0 and accept H_1 ie. Job satisfaction and pension plan are Associated.

χ^2 – Test of Goodness of fit

It is a test of fitness between observed and expected frequencies. Here –

H_0 : The fit is good between observed (O) and expected frequencies (E)

ie. $O \approx E$

H_1 : The fit is not good.

The test statistic is $\chi^2 = \Sigma(O - E)^2 / E \sim \chi^2_{(k-1)\text{ df.}}$ Here E are found as stated in the problem. If Calculated $\chi^2 \leq$ Critical $\chi^2_{(k-1)\text{df}}$ at α LOS then H_0 is accepted otherwise H_0 is rejected.

Ex 16 **:** Following is the record of number of accidents which took place during the vatious days of the week in India.

Days	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
No. of Accidents	184	148	145	153	150	154	116

Test whether accidents are uniformly distributed over the entire week at 5% LOS.

Soln : H_0 **:** Accidents are uniformly distributed.

H_1 **:** Accidents are not uniformly distributed.

Then we take E = mean of accidents for each day.

$$= (184 + 148 + 145 + 153 + 150 + 154 + 116) / 7 = 150$$

O	E	$(O - E)^2$	$(O - E)^2/E$
184	150	1156	7.71
148	150	4	0.03
145	150	25	0.17
153	150	9	0.06
150	150	0	0
154	150	16	0.11
116	150	1156	7.71
		Chi-square	15.79

Practice Problems

Q1. A sample of 25 sales receipts from a grocery store has average Rs. 950 and variance = 400. Use these values to test whether or not the mean is sales at the grocery store are different from Rs. 960.

Q2. An insurance company is reviewing its current policy rates. When originally setting the rates they believed that the average claim amount was \$1,800. They are concerned that the true mean is actually higher than this, because they could potentially lose a lot of money. They randomly select 40 claims, and calculate a sample mean of \$1,950. Assuming that the standard deviation of claims is \$500, and set $\alpha = 0.05$ test to see if the insurance company should be concerned.

Q3. From long experience with a process for manufacturing an alcoholic beverage it is known that the yield is normally distributed with a mean of 500 and a standard deviation of 96 units . For a modified process the yield is 535 units for a sample of size 50. At $\alpha = .05$ does the modified process increase the yield?

Practice Problems

Q4. Trying to encourage people to stop driving to campus, the university claims that on average it takes people 30 minutes to find a parking space on campus. I don’t think it takes so long to find a spot. In fact I have a sample of the last five times I drove to campus, and I calculated $x = 20$. Assuming that the time it takes to find a parking spot is normal, and that $\sigma = 6$ minutes, then perform a hypothesis test with level $\alpha = 0.10$ (ie. 10%) to see if my claim is correct.

Q5. Average IQ of the students of an Institute was stated to be utmost 130 with a variance of 144. A sample of 25 students were tested and their IQ was found to be 125.
Test the significance of the statement at a confidence level of 95%.

Q6. A a sample of size 9 from a normal population is given below. Find the 90% confidence interval (CI) for the mean μ of the population. Also find the 90% confidence interval for the variance σ^2 of the population. 0, 1, -1, 1, 1, 0, -1, -2, 3.

Practice Problems

Q7. A car manufacturing company claims that their cars do not require servicing before running for on an average 10,000 miles. To verify the claim, the manufacturer tests 30 cars under simulated conditions and gets a mean breakdown of 9500 miles with a standard deviation of 1150 miles. What conclusion can be drawn at a significance level of 5%.

Q8. Find the probabilities that a random variable having standard normal distribution will take on a value

(a)Between 0.87 and 1.28
(b)Greater than 0.85
(c)Between -0.34 and 0.62;

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

BSDCHZC355
Statistical Inference & Applications

Shaibal Kr. Sen
Session 03

innovate achieve lead

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

STATISTICAL INFERENCES

&

APPLICATIONS

Scheduled Topics to be covered

Introduction to **D**esign of **E**xperiments (DOE)

Basic **P**rinciples & **G**uidelines of **DOE**

Inferences about **D**ifferences in **M**eans & **R**andomised **D**esigns

What is Design of Experiments (DOE)?

Design means **d**eveloping a method or procedure or process or way.

Experiment means a Way or Method or Procedure or Process of obtaining an answer to a real life problem.

Design of **E**xperiments means developing a method to obtain valid inference (conclusion or answer) to a problem.

Statistical design of experiments refers to the process of planning the experiment so that appropriate data will be collected and analyzed by statistical methods, resulting in valid and objective conclusions.

The statistical approach to experimental design is necessary if we wish to draw meaningful conclusions from the data.

What is Design of Experiments (DOE)?

When the problem involves data that are subject to experimental errors, statistical methods are the only objective approach to analyse.

Thus, there are two aspects to any experimental problem –

Design of the Experiment and

Statistical Analysis of the data.

These two subjects are closely related because the method of analysis depends directly on the design employed.

Terminology

Before discussing the principles of designs, it is proper to explain the terminology used in this context. The terms commonly used are – **E**xperiment, **T**reatment, **E**xperimental **U**nit, **E**xperimental **E**rror and **P**recision.

Experiment **:** It is a means of getting an answer to the question that the experimenter has in mind. This may be to decide which of several pain-relieving drugs that are available in the market is the most effective or whether they are equally effective. An experiment may be planned to compare the Chinese method of cultivation with the standard method used in India. In planning an experiment, we clearly state our objectives & formulate the hypotheses we want to test.

Treatment **:** The different procedures under comparison in an experiment are the different treatments eg. – in an agricultural experiment, the different varieties of a crop or the different manures will be the treatments. In a dietary or medical experiment, the different diets or medicines, etc. are the treatments.

Experimental **U**nits **:** In carrying out an experiment, we should be clear as to what constitutes the experimental unit. An experimental unit is the material to which is applied the treatment and on which the variable under study is measured. In an agricultural field experiment, the plot of land and not the individual plant, will be the experimental unit; in a feeding experiment of cows, the whole cow is the experimental unit; in human experiments in which the treatment affects the individual, the individual will be the experimental unit.

Experimental **E**rror **:** A fundamental phenomenon in replicated experiments is the variation in the measurements made on different experimental units even when they get the same treatment. A part of this variation is systematic and can be explained, whereas the remainder is to be taken to be of the random type. The unexplained random part of the variation is termed the experimental error. This is a technical term and does not mean a mistake, but includes all types of extraneous (irrelevant or unrelated to the subject) variation due to –

(i) Inherent variability in the experimental units.

(ii) Errors associated with the measurements made.

(iii) Lack of representativeness of the sample to the population under study.

The experimental error provides a basis for the confidence to be placed in the inference about the population. So it is important to estimate & control the experimental error. An estimate of the experimental error can only be obtained by replication, and it is controlled by the principle of local control, to be explained shortly.

The precision of an experiment is measured by the reciprocal of the variance of a mean -

$1 / (\sigma)^2 / n = n / \sigma^2$

As n (replication number) increases, precision also increases. Another means of increasing precision is to control σ^2. The smaller the value of σ^2, the greater the precision.

Whether in Engineering, R&D, or Science Lab, understanding the basics of experimental design can help to achieve more statistically optimal results applying experimental techniques and improve output quality.

Using Design of Experiments (DOE) techniques, one can determine the individual and interactive effects of various factors that can influence the output results of measurements. It can also be used to gain knowledge and estimate the best operating conditions of a System, Process or Product.

DOE applies to many different investigation objectives, but can be especially important early on in a screening investigation to help in determining the most important factors which can influence the responses or critical quality attributes.

Basic Principles of DOE

Basic Principles of DOE are – Randomization, Replication, Blocking, Multifactor Designs and Confounding

The first three here are perhaps the most important –

Randomization **:** By randomization we mean that both the allocation of the experimental material and the order in which the individual runs of the experiment are to be performed are randomly determined.

Statistical methods require that the observations (or errors) be independently distributed random variables. Randomization usually makes this assumption valid.

By properly randomizing the experiment, we also assist in "averaging out" the effects of extraneous (unnecessary-unwanted-untargeted for study) factors that may be present. It will help to eliminate potential biases from the conclusions.

Replication **:** By replication we mean an independent repeat run of each factor combination.

Replication has two important properties –

First, it allows the experimenter to obtain an estimate of the experimental error.

Second, it permits the experimenter to obtain a more precise estimate of the parameter.

***B**locking* (Local Control) **:** Blocking is a design technique used to improve the precision with which comparisons among the factors of primary interest are made.

Often blocking is used to reduce or eliminate the variability transmitted from nuisance (extraneous) factors - that is, factors that may influence the experimental response but in which we are not directly interested.

Generally, a block is a set of relatively homogeneous experimental conditions and the variability within a block would be expected to be smaller than the variability between blocks.

The experimenter divides the observations from the statistical design into groups that are run in each block.

Multi-Factor Designs **:** These designs are meant to evaluate multiple factors set at multiple levels. One approach is called a Full Factorial experiment, in which each factor is tested at each level in every possible combination with the other factors and their levels. Full factorial experiments that study all paired interactions can be economic and practical if there are few factors and only 2 or 3 levels per factor. The advantage is that all paired interactions can be studied. However, the number of runs goes up exponentially as additional factors are added. Experiments with many factors can quickly become unwieldy and costly to execute, as shown by the chart in next slide. To avoid complexity, factors which are having major impact on the response variable should taken and factor os minor significance should be avoided.

Experimental work when many factors are likely to be investigated. It

Relationship of Experimental Run with Factor-Level-Replicate

provides the smallest number of runs with which *k* factors can be studied in a complete factorial design. Consequently, these designs are widely used in *factor screening experiment.*

Relationship between Experimental Run and Factor-Level-Replicate

No. of Factors (F)	No. of Levels per Factor (L)	No. of Replicates (R)	No. of Runs Full Factorial $(L^F) \times R$
2	2	1	(2^2)x1 = 4
2	3	2	(3^2)x2 = 18
3	2	2	(2^3)x2 = 16
3	3	3	(3^3)x3 = 81
4	2	2	(2^4)x2 = 32
4	3	4	(3^4)x4 = 324
5	2	3	(2^5)x3 = 96
5	3	4	(3^5)x4 = 972
6	2	3	(2^6)x3 = 192
6	3	5	(3^6)x5 = 3645

Confounding - is something that is usually considered bad! Here is an example. Let's say we are doing a medical study with drugs X and Y. We put 10 subjects on drug X and 10 on drug Y. If we categorize our subjects by gender, how should we allocate our drugs to our subjects? Let's make it easy and say that there are 10 male and 10 female subjects. A balanced way of doing this study would be to put five males on drug X and five males on drugY, five females on drug X and five females on drug Y. This is a perfectly balanced experiment such that if there is a difference between male and female at least it will equally influence the results from drug X and the results from drug Y.

An alternative scenario might occur if patients were randomly assigned treatments as they came in the door. At the end of the study they might realize that drug X had only been given to the male subjects and drug Y was only given to the female subjects. We would call this design totally confounded. This refers to the fact that if you analyze the difference between the average response of the subjects on X and the average response of the subjects on Y, this is exactly the same as the average response on males and the average response on females. You would not have any reliable conclusion from this study at all. The difference between the two drugs X and Y, might just as well be due to the gender of the subjects, since the two factors are totally confounded.

Confounding is something we typically want to avoid but when we are building complex experiments we sometimes can use confounding to our advantage. We will confound things we are not interested in order to have more efficient experiments for the things we are interested in. This will come up in multiple factor Experiments. We may be interested in main effects but not interactions so we will confound the interactions in this way in order to reduce the sample size, and thus the cost of the experiment, but still have good information on the main effects.

Key Benefits of DOE

1. Optimization of Processes **:** Helps in identifying the most influential factors in a process and determining the optimal settings for those factors. This leads to improved efficiency and performance, as processes can be fine-tuned based on empirical data.

2. Efficient Utilization of Resource **:** By designing experiments to collect the most relevant data with fewer trials, DOE minimizes the number of experiments needed. This saves time, reduces costs, and uses fewer resources.

3. Improved Product Quality **:** With DOE, it's possible to identify the factors that most significantly impact product quality. By controlling and optimizing these factors, organizations can produce higher-quality products more consistently.

4. Data-Driven Decisions **:** DOE uses statistical methods to analyze data

and draw conclusions, reducing the reliance on guesswork or intuition. This ensures that decisions are based on solid data and not on assumptions or biases.

5. Identifying Interactions Between Variables **:** One of the key strengths of DOE is its ability to detect interactions between different factors. Understanding these interactions helps in making more informed decisions about how multiple variables affect outcomes.

6. Reduction in Variability **:** By understanding the relationship between factors and their influence on variability, DOE helps reduce unwanted variation, leading to more stable and predictable processes.

7. Better Risk Management **:** DOE allows you to predict and control the impact of various factors on the final outcome, helping to identify and mitigate risks early in the process or product development.

8. Improved Decision-Making **:** The insights gained from DOE experiments provide a clearer understanding of cause-and-effect relationships, allowing for better decisions in product development, manufacturing, and other areas.

9. Flexibility **:** DOE can be applied across various industries and disciplines, from manufacturing and engineering to healthcare, marketing and agriculture, making it a versatile tool for experimentation and improvement.

10. Scalability **:** DOE is not only useful in small-scale experiments but can also be scaled up to large, complex systems. As businesses grow, DOE helps scale their processes while maintaining quality and efficiency.

In summary, the benefits of Design of Experiments lie in its ability to optimize processes, improve decision-making, enhance product quality, and reduce variability - all while using resources efficiently.

Already covered in CS2

Z test of difference of two means

For testing the significant difference between two population means using large samples ($n \geq 30$), the test statistic is -

$Z = (\overline{x}_1 - \overline{x}_2) / [\sqrt{\{(\sigma_1)^2/n_1 + (\sigma_2)^2/n_2\}}] = (\overline{x}_1 - \overline{x}_2) / [\sqrt{\{(s_1)^2/n_1 + (s_2)^2/n_2\}}]$

and the significant value is Z_α for one tailed test and $Z_{\alpha/2}$ for two tailed test.

μ_1 and μ_2 are two population means respectively.

$(\sigma_1)^2$ and $(\sigma_2)^2$ are two population variances respectively and if they are not known they are replaced by their respective sample variances $(s_1)^2$ and $(s_2)^2$.

n_1 and n_2 are two sample sizes respectively (each ≥ 30)

Here, $H_0 : \mu_1 = \mu_2$ and $H_1 : \mu_1 \neq \mu_2$

If the calculated $|Z| \leq$ significant value (obtained from Z-table), then H_0 is accepted; otherwise H_0 is rejected.

Testing the difference between means using t-test

Assumptions : Two independent populations follow (i) **N**ormality with (ii) Generally with **E**qual unknown **V**ariances (s^2). (iii) The two random **S**ample **S**izes (n_1 and n_2) are less than 30. The averages of sample 1 & sample 2 are respectively $\overline{x}_1$ & $\overline{x}_2$.

Algorithm :

Step 1 - State Null and Alternative Hypothesis -

$H_0 : \mu_1 = \mu_2$ vs $H_1 : \mu_1 < \mu_2$ or, $\mu_1 > \mu_2$ or, $\mu_1 \neq \mu_2$

Step 2 - Specify the level of significance 'α'

Step 3 - Assuming Standard Normal Distribution compute test statistic 't'.

Test Statistic $t = (\overline{x}_1 - \overline{x}_2) / [s\sqrt{\{1/n_1 + 1/n_2\}}]$

where, $s^2 = [(n_1 - 1)(s_1)^2 + (n_2 - 1)(s_2)^2 / (n_1 + n_2 - 2)$ (if $s_1 \neq s_2$)

Step 4 – Find the t-value from t-table for $t_{\alpha,df}$ & Define the critical region ie. Acceptance & Rejection criteria. Here, df ie. degree of freedom = $n_1 + n_2 - 2$

Step 5 – Draw inference whether to Accept or Reject the Hypothesis based on critical value or P-value.

Ex 1 **:** The manager of a courier service believes that packets delivered at the beginning of the month are heavier than those delivered at the end of month. As an experiment, he weighed a random sample of 15 packets at the beginning of the month and found that the mean weight was 5.25 kg. A randomly selected 10 packets at the end of the month had a mean weight of 4.26 kg. It was observed from the past experience that the sample variances are 1.20 kg^2 and 1.15 kg^2. Determine -

(i) At 5% level of significance, can it be concluded that the packets delivered at the beginning of the month weigh more?

(ii) Also find P-value.

Soln **:** Given – $\bar{x}_1 = 5.25$, $\bar{x}_2 = 4.56$, $n_1 = 15$, $n_2 = 10$, $(s_1)^2 = 1.2$, $(s_2)^2 = 1.15$

Degree of freedom = $n_1 + n_2 - 2 = 15 + 10 - 2 = 23$; $t_{0.05,\,23} = 1.714$

(from t-table).

$s = \sqrt{[(n_1 - 1)(s_1)^2 + (n_2 - 1)(s_2)^2 / (n_1 + n_2 - 2)]} = \sqrt{[(14x1.2 + 9x1.15) / 23]}$

= 1.086

(i) $H_0 : \mu_1 - \mu_2 = 0$ and $H_1 : \mu_1 - \mu_2 \neq 0$

Test statistic $t = (\overline{x}_1 - \overline{x}_2) / [s\sqrt{\{1/n_1 + 1/n_2\}}]$

$= (5.25 - 4.26) / [1.086\sqrt{(1/15 + 1/10)}] = 2.234$

Since, t_{stat} (2.234) > $t_{critical}$ (1.714) reject H_0.

(ii) P- value corresponding to t = 2.234 is $0.01 < P < 0.025$ ie. less than 0.05.

Ex 2 : As a manager of finance you are assigned the task of choosing the best product in terms of life of the product. Use 5% level of significance.

Product A - Mean 1456 hours with SD 423, size 10

Product B - Mean 1280 hours with SD 398, size 17

Soln **:** $H_0 : \mu_1 = \mu_2$ and $H_1 : \mu_1 \neq \mu_2$

$s^2 = (n_1 - 1)s_1^2 + (n_2 - 1)s_2^2 / (n_1 + n_2 - 2) = 9(423)^2 + 16(398)^2 / (10 + 17 - 2)$

$= 1,65,793 \qquad s = 407.18$

Test stat $t = \{(\bar{x}_1 - \bar{x}_2) - (\mu_1 - \mu_2)\} / \{s\sqrt{(1/n_1 + 1/n_2)}\}$

$= \{(1456 - 1280) - 0\} / \{407.18(1/10 + 1/17)\} = 1.085$

Degree of Freedom $= n_1 + n_2 - 2 = 10 + 17 - 2 = 25$

We obtain from t-table, $t_{0.05,\ 25} = 2.06$

Since test statistic t falls within acceptance region, we accept the null hypothesis.

Ex 3 : Random samples of 15 and 10 were selected from two thermocouples. The sample means were 315, 303 and sample standard deviations were 3.8, 4.9 respectively. Test whether there is any significant difference in the means of two thermocouples at 5% level of significance

Soln : Given $\bar{x}_1 = 315$, $\bar{x}_2 = 303$, $s_1 = 3.8$, $s_2 = 4.9$, $n_1 = 15$, $n_2 = 10$

LOS $\alpha = 0.05$, Degree of freedom = $n_1 + n_2 - 2 = 15 + 10 - 2 = 23$

$t_{0.05,\,23} = 1.714$ (From t-table)

$H_0 : \mu_1 = \mu_2$ and $H_1 : \mu_1 \neq \mu_2$

Test statistic $t = [(\bar{x}_1 - \bar{x}_2) - (\mu_1 - \mu_2)] / [\{\sqrt{(s_1)^2/n_1 + (s_2)^2/n_2}\}]$

$= (315 - 303) / [\{\sqrt{(3.8)^2/15 + (4.9)^2/10}\}] = 12 / 1.834 = 6.54$

Test statistics value is beyond acceptance region, so we can not accept the null hypothesis.

Ex 4 : The researchers reported the following sample statistics.
In a sample of 45 women dining with other women, the average number of calories ordered was 850, and the standard deviation was 252.
In a sample of 27 women dining with men, the average number of calories ordered was 719, and the standard deviation was 322.
(i) At 5% LOS can we conclude that average calories ordered by both the group of women are same?
(ii) Also evaluate the P-value.

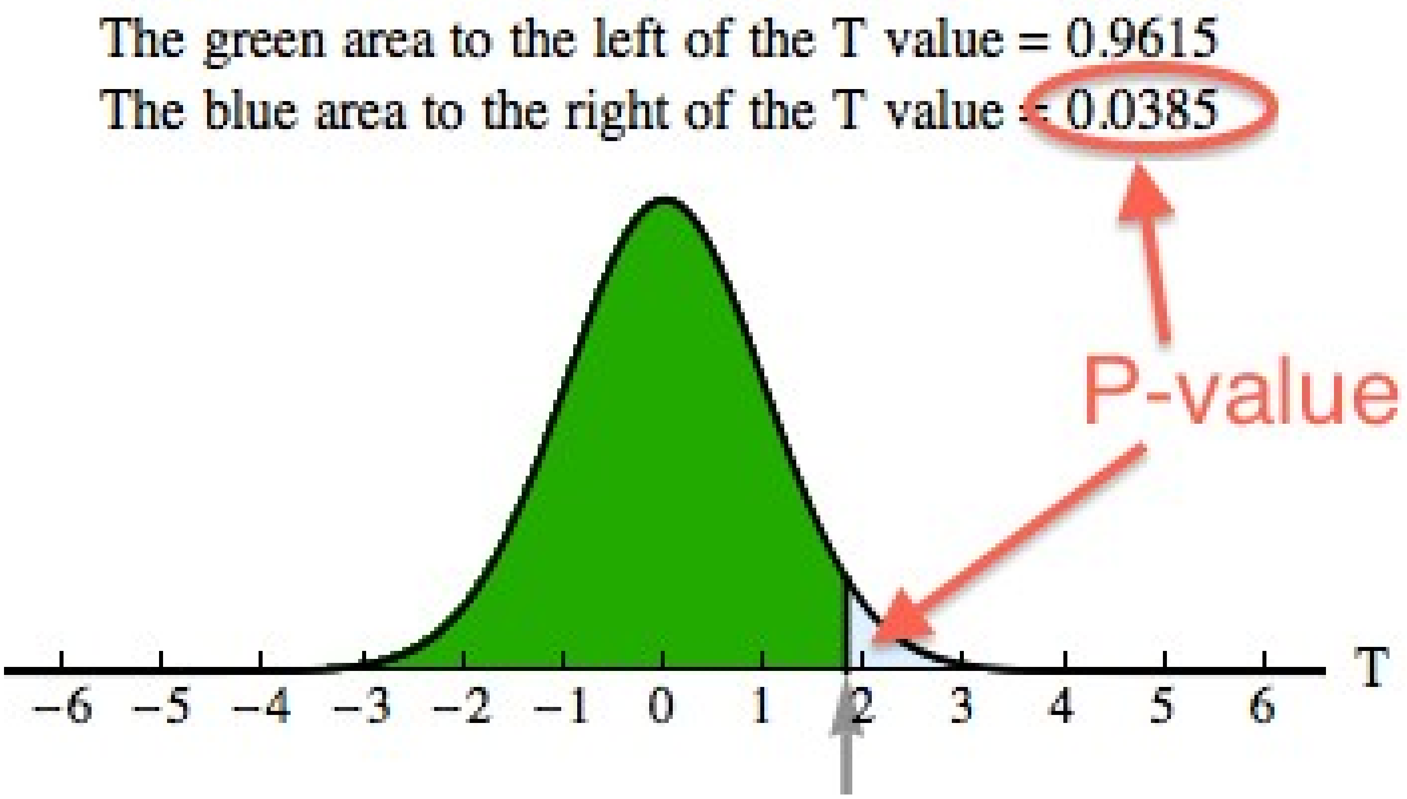
Soln : Using the steps applied in previous numerical we proceed to find

(i) $t_{stat} = (850 - 719) / [\sqrt{(252)^2 / 45 + (322)^2 / 27}] \approx 131 / 72.47 \approx 1.81$

$t_{critical} = 1.645$

Since, $t_{stat} > t_{critical}$, null hypothesis is rejected.

(ii)



The green area to the left of the T value = 0.9615

The blue area to the right of the T value = 0.0385

1.81

P-value = 0.0385 which is less than the significance level 0.05. Hence, we reject null hypothesis and accept alternative hypothesis.

Practice Problems

Q1. Explain usefulness of Design of Experiment (DOE)?

Q2. Random samples of 25 and 20 were selected from two thermocouples. The sample means were 321, 308 and sample standard deviations were 4.1, 4.7 respectively. Test whether there is any significant difference in the means of two thermocouples at 1% level of significance

Q3. Is it true that experiments enhance performance of Production Process? Justify your answer.

Q4. State the salient features of DOE and explain two important ones.

Q5. Explain Assignable causes of Quality Variation.

Q7. Narrate the Experimental Design Process.

Practice Problems

Q8. There is an assumption that there is no significant difference in the productivity of men and women workers in a production center. Tests were conducted on two groups and the following are the observations.

	Sample Size	Average Productivity	SD
Men	50	100	9
Women	60	95	8

Validate this at 5% Level of Significance.

Q9. Fill-up the blank column –

No. of Factors	No. of levels	No. of runs for full Factorial Design
2	2	
2	5	
3	3	
4	3	
5	4	

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

BSDCHZC355
Statistical Inference & Applications

Shaibal Kr. Sen
Session 04

innovate achieve lead

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

STATISTICAL INFERENCES

&

APPLICATIONS

Scheduled Topics to be covered

Experiment with a **S**ingle **F**actor **: A**nalysis of **V**ariance (**A**NOVA)

ANOVA **I**ntroduction

One **W**ay **A**NOVA **C**lassification

Analysis of Variance (ANOVA)

When do you use an ANOVA?

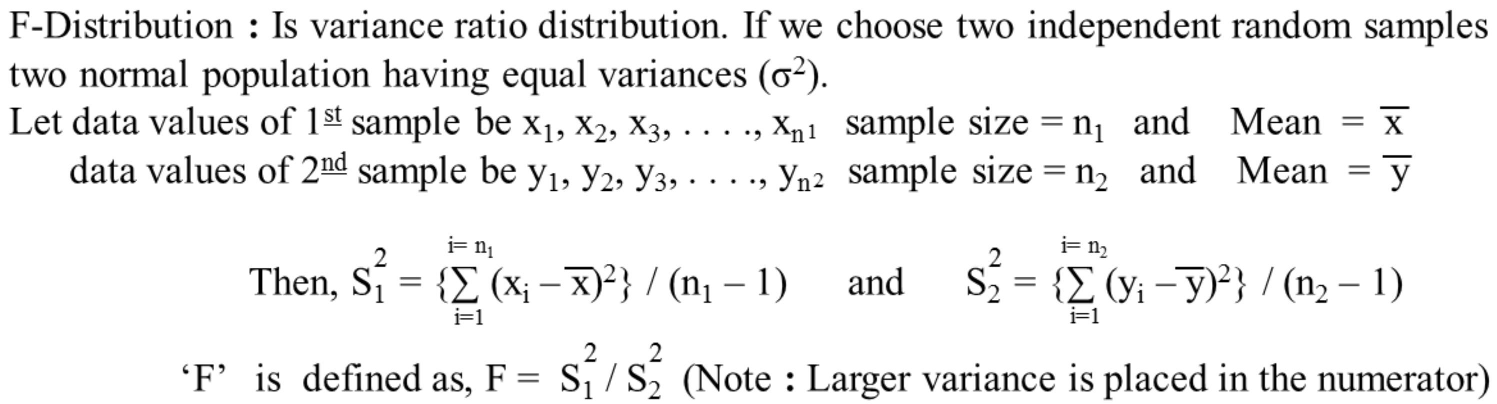
ANOVA is used to determine whether there are any significant differences between the means of three or more independent (unrelated) groups. Here we will provide a brief introduction to the one-way ANOVA, including the assumptions of the test and when you should use this test.

Analysis of variance (ANOVA) is a collection of statistical models used in order to analyze the differences among group means and their associated procedures (such as “variation” among and between groups), developed by statistician and evolutionary biologist Ronald Fisher. In the ANOVA setting, the observed variance in a particular variable is partitioned into components attributable to different sources of variation. In its simplest form, ANOVA provides a statistical test of whether or not the means of several groups are equal, and therefore generalizes the *t*-test to more than two groups. As doing multiple two-sample t-tests would result in an increased chance of committing a statistical type I error, ANOVAs are useful in comparing (testing) three or more means (groups or variables) for statistical significance.

Analysis of variance seeks to identify sources of variation in numerical dependent variable Y (the response variable). Variation in the response variable about its mean either is explained by one or more categorically independent variable/s (the factors) or is unexplained (random error).

Variation in Y = Explained Variation + Unexplained Variation
(Around its Mean) (Due to Factors) (Random Error)

ANOVA is a comparison of means. Each possible value of a factor or combination of factors is a treatment. Sample observations within each treatment are viewed as coming from populations with possibly different means. We test whether each factor has a significant effect on Y, and sometimes we test for interaction between factors. The test uses the F distribution.



F-Distribution : Is variance ratio distribution. If we choose two independent random samples from two normal population having equal variances (σ^2).

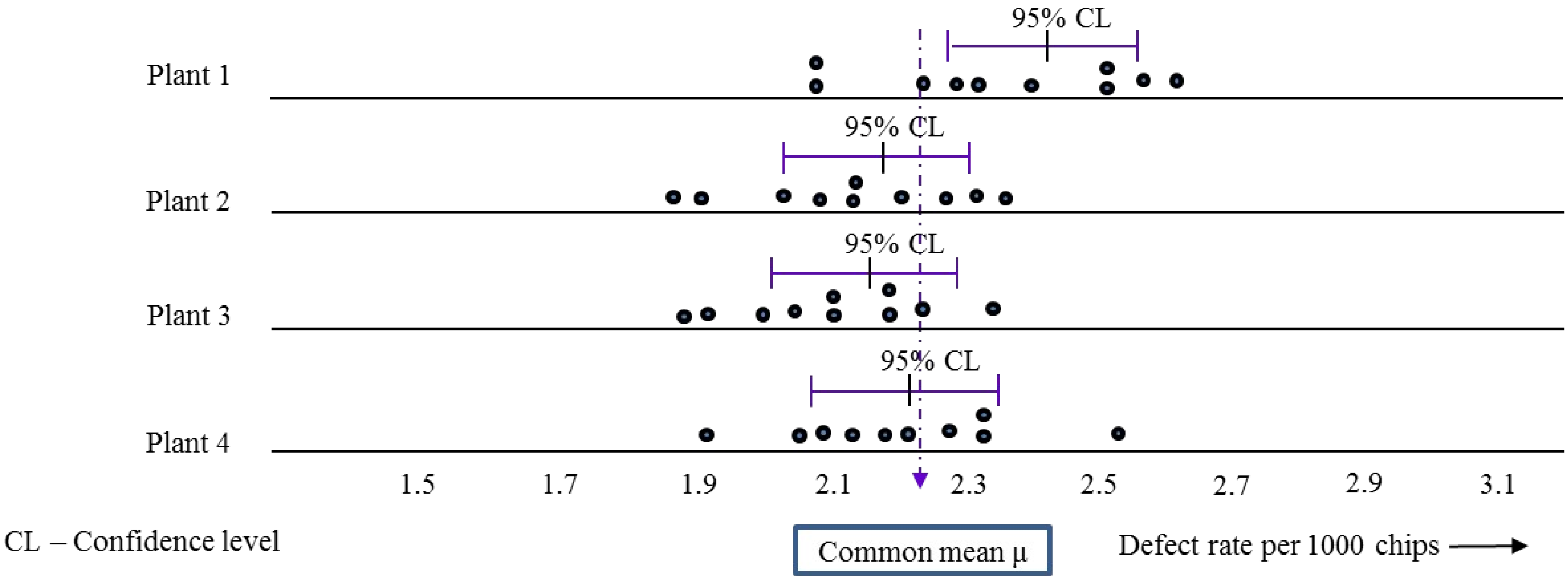
Let data values of 1st sample be $x_1, x_2, x_3, \ldots, x_{n1}$ sample size = n_1 and Mean = $\overline{x}$
data values of 2nd sample be $y_1, y_2, y_3, \ldots, y_{n2}$ sample size = n_2 and Mean = $\overline{y}$

$$\text{Then, } S_1^2 = \{\sum_{i=1}^{i=n_1} (x_i - \overline{x})^2\} / (n_1 - 1) \quad \text{and} \quad S_2^2 = \{\sum_{i=1}^{i=n_2} (y_i - \overline{y})^2\} / (n_2 - 1)$$

'F' is defined as, $F = S_1^2 / S_2^2$ (Note : Larger variance is placed in the numerator)

ANOVA can handle any number of factors, but the researcher often is interested only in a few. Also, data collection costs may impose practical limits on the number of factors or treatments we can choose. We will focus on ANOVA with one or two factors.

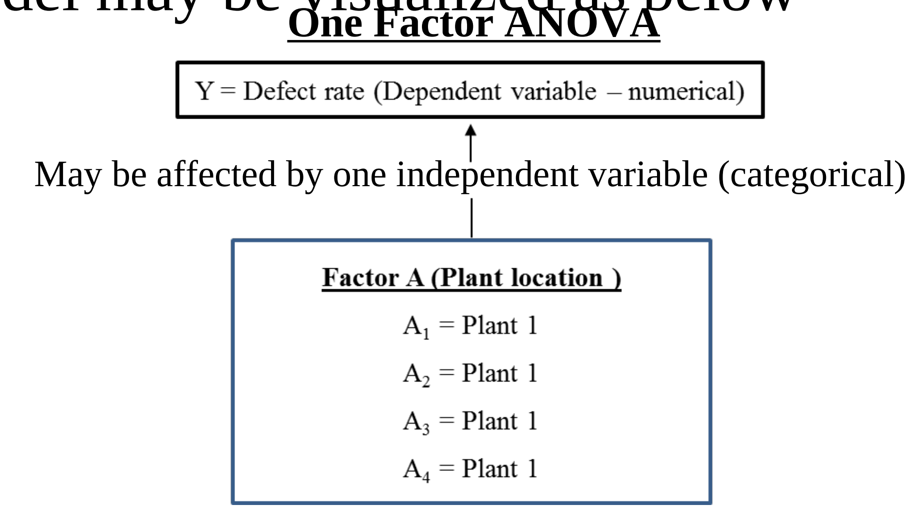
Manufacturing defect rates **:** Figure below shows a dot plot of daily defect rates for automotive computer chips mfgd at four plant locations. Samples of 10 days production were taken at each plant. Are the observed differences in the plants' sample mean defect rates merely due to random variation? Or are the observed differences between the plants' defect rates too great to be attributed to chance? This is the kind of question that ANOVA is designed to answer.


95% CL
Plant 1
95% CL
Plant 2
95% CL
Plant 3
95% CL
Plant 4
1.5
1.7
1.9
2.1
2.3
2.5
2.7
2.9
3.1
CL – Confidence level
Common mean μ
Defect rate per 1000 chips →

A simple way to state the ANOVA hypothesis is

$H_0 : \mu_1 = \mu_2 = \mu_3 = \mu_4$ (mean defect rates are same at all four plants)
H_1 : Not all the means are equal (at least one mean differs from the others)

If we can not reject H_0, then we conclude that the observations within each treatment or group actually have a common mean μ (represented by dotted line) in the previous slide. This one factor ANOVA model may be visualized as below –



One Factor ANOVA

Y = Defect rate (Dependent variable – numerical)

May be affected by one independent variable (categorical)

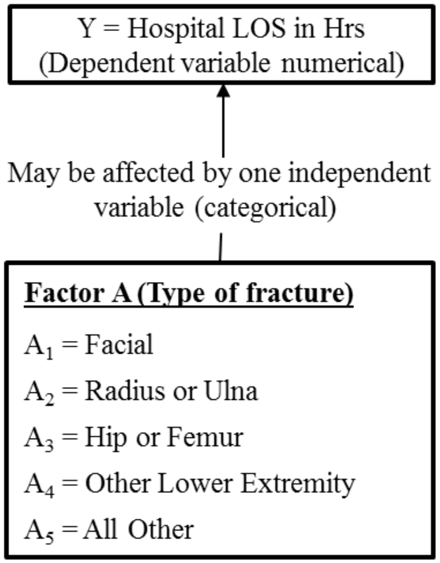
Factor A (Plant location)

A_1 = Plant 1
A_2 = Plant 1
A_3 = Plant 1
A_4 = Plant 1

Hospital length of stay

To allocate resources and fix costs correctly, hospital management needs to test whether a patient’s length of stay (LOS) depends on the diagnostic related grp (DRG) code and the patient’s age grp. Consider the case of a bone fracture. LOS is a numerical response variable (measured in hours). The hospital organizes the data by using five diagnostic codes for age grps (under 18, 18 to 64 & 65 and above). Although patient age is a numerical variable, it is coded into 3 categories based on stages of bone growth. Figure below illustrates two possible ANOVA models (one factor and two factor). We could also test for interaction between factors, as you will see later on.

One Factor ANOVA



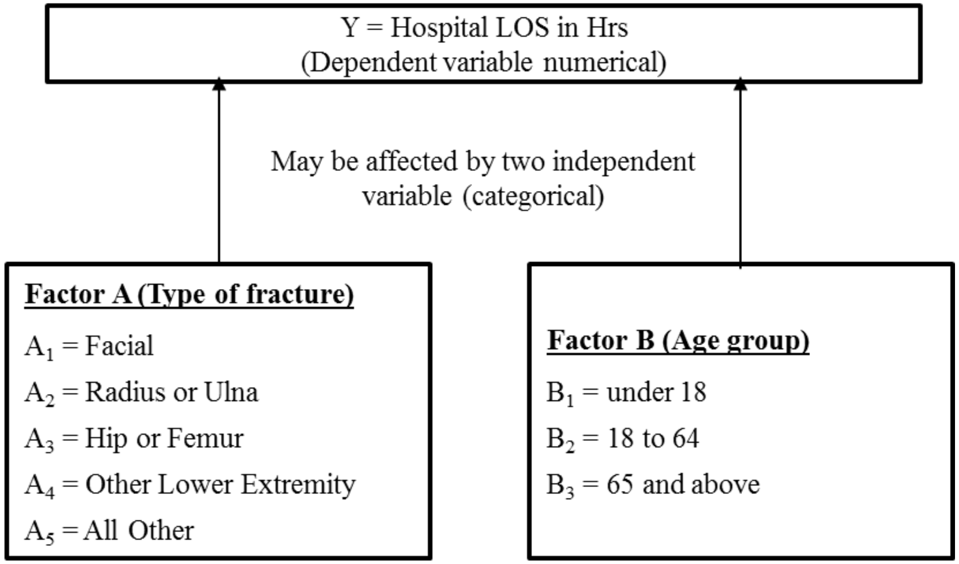
Y = Hospital LOS in Hrs
(Dependent variable numerical)

May be affected by one independent variable (categorical)

Factor A (Type of fracture)

A_1 = Facial
A_2 = Radius or Ulna
A_3 = Hip or Femur
A_4 = Other Lower Extremity
A_5 = All Other

Two Factor ANOVA



Y = Hospital LOS in Hrs
(Dependent variable numerical)

May be affected by two independent variable (categorical)

Factor A (Type of fracture)

A_1 = Facial
A_2 = Radius or Ulna
A_3 = Hip or Femur
A_4 = Other Lower Extremity
A_5 = All Other

Factor B (Age group)

B_1 = under 18
B_2 = 18 to 64
B_3 = 65 and above

Similarly, we can design various ANOVA examples for more such cases viz. – Automobile Paintings (Different paint suppliers eg. – Asian Paints, Berger, Jenson & Nicholson etc.), Car Manufacturing (Maruti, Honda, Hyundai etc. etc.), FMCG Regional Distributor (Eastern, Western, Northern and Southern workstations) services etc. etc.

ANOVA calculations are usually too tedious to do by calculator, so after we choose an ANOVA model and collect the data, we rely on software [eg. Excel, Megastat, Minitab, SPSS (Statistical Package for the Social Sciences)] to do the calculations. In some applications viz. – Accounting, Finance, HR, Marketing large samples can easily be taken from existing records, while in others viz. – Engg, Mfg, Computer Systems experimental data collection is so expensive that small samples are used. Large samples increase the power of the test, but power also depend on degree of variation in Y. Lowest power would be in small samples with higher variation in Y, and conversely. Specialised software is needed to calculate power for ANOVA experiments.

ANOVA Assumptions

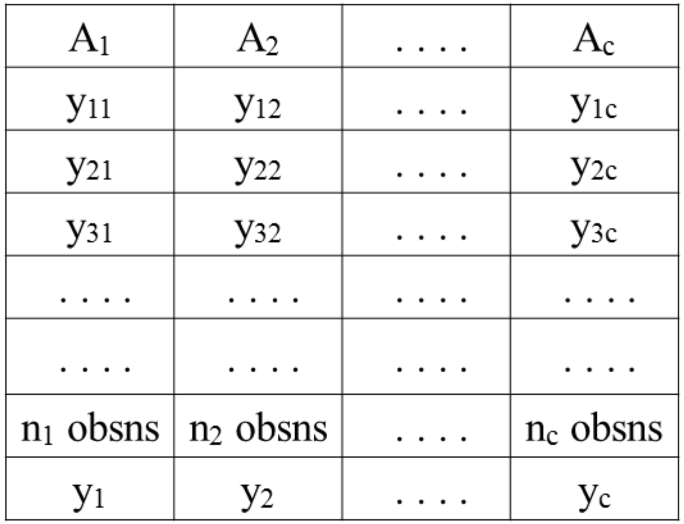
Observations in Y are independent.
Populations being sampled are normal.
Populations being sampled have equal variances.

Fortunately, ANOVA is somewhat robust to departures from the normality and equal variance assumptions.

One factor ANOVA (Completely randomized model) **:** Data format - In the following illustration, if we are only interested in comparing of C-grps (treatments or factor levels), we have a one factor ANOVA. This is by far the most common ANOVA model that covers many business problems. The one factor ANOVA is usually viewed as a comparison between several columns of data, although the data could also be presented in rows.

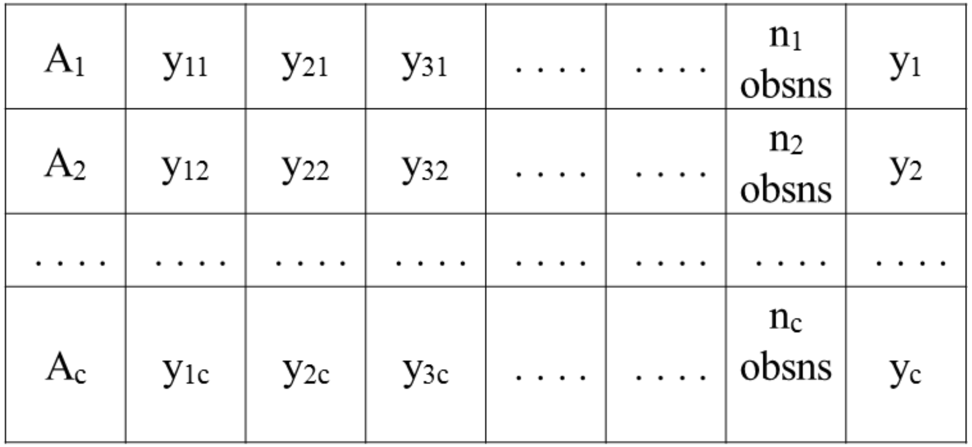
Table below illustrates the data format for a one-factor ANOVA with C treatments, denoted A_1, A_2, A_3, ... , A_c. The group means are y_1, y_2, y_3, ... , y_c.

One-factor ANOVA data in column



A_1	A_2	. . . .	A_c
y_{11}	y_{12}	. . . .	y_{1c}
y_{21}	y_{22}	. . . .	y_{2c}
y_{31}	y_{32}	. . . .	y_{3c}
. . . .	. . . .	. . . .	. . . .
. . . .	. . . .	. . . .	. . . .
n_1 obsns	n_2 obsns	. . . .	n_c obsns
y_1	y_2	. . . .	y_c

One-factor ANOVA data in rows



A_1	y_{11}	y_{21}	y_{31}	. . . .	. . . .	n_1 obsns	y_1
A_2	y_{12}	y_{22}	y_{32}	. . . .	. . . .	n_2 obsns	y_2
. . . .	. . . .	. . . .	. . . .	. . . .	. . . .	. . . .	. . . .
A_c	y_{1c}	y_{2c}	y_{3c}	. . . .	. . . .	n_c obsns	y_c

Within treatment j we have n_j observations on y. Sample sizes within each treatment do not need to be equal, although there are

advantages to having balanced sample sizes. The total number of observations is the sum of the sample sizes for each treatment

$$n = n_1 + n_2 + n_3 + \dots + n_c$$

Hypothesis to be tested **:** The question of interest is whether the mean of Y varies from treatment to treatment. The hypothesis to be tested are –

Null Hypothesis H_0 **:** $\mu_1 = \mu_2 = \mu_3 = \dots \mu_c$ (ie. all the means are equal)

Alternative Hypo H_1 **:** Not all the means are equal (at least one mean is different)

Since one-factor ANOVA is a generalization of the test for equality of two means, why not just compare all possible pairs of means by using repeated two sample t-test. Consider our experiment comparing the four mfg plant average defect rates. To compare pairs of plant averages we would have to perform six (${}^{4}C_{2}$) different t-tests. If each t-test has a Type I error probability equal to 0.05, then the probability that at least one of these test results in a Type I error is $1 – (0.95)^6 = 0.2649$. ANOVA tests all the means simultaneously and therefore does not inflate our Type I error.

One factor ANOVA as a linear model **:** An equivalent way to express the one factor model is to say that observations in treatment j came from a population with a common mean (μ) plus a treatment effect (A_j) plus random error (e_{ij}) :

$y_{ij} = \mu + A_j + e_{ij}$ where, $j = 1, 2, ... , c$ & $i = 1, 2, ... , n_j$

The random error is assumed to be normally distributed with zero mean and same variance for all treatments. If we are interested only in what happens to the response for the particular levels of the factor that were selected (a fixed-effect model), then the hypothesis to be tested are –

H_0 : $A_1 = A_2 = ... = A_c = 0$ (all treatment effects are zero)

H_1 : Not all A_j are zero (some treatment effects are non-zero)

If the null hypothesis is true ($A_j = 0$ for all j), then knowing that an observation x came from treatment j does not help explain the variation in Y and the ANOVA model collapses to

$$y_{ij} = \mu + e_{ij}$$

Group means : The mean of each group is calculated in the usual way by summing the observations in the treatment and dividing by the sample size :

$$\overline{y_j} = (1 / n_j)\sum_{i=1}^{n_j} y_{ij}$$

The overall sample mean or grand mean y can be calculated either by summing all the observations and dividing by n or by taking a weighted average of the c sample means :

$$\overline{y} = (1 / n)\sum_{j=1}^{c} \sum_{i=1}^{n_j} y_{ij} = (1 / n)\sum_{j=1}^{c} n_j y_j$$

Partitioned sum of squares : To understand the logic of ANOVA, consider that for a given observation y_{ij} the following relationship must hold (on the RHS we just add & subtract $\overline{y_j}$) –

$$(y_{ij} - \overline{y}) = (\overline{y_j} - \overline{y}) + (y_{ij} - \overline{y_j})$$

This says that any deviation of an observation from the grand mean $\bar{y}$ may be expressed in two parts - deviation of column mean ($\bar{y}_j$) from the grand mean ($\bar{y}$) , or between treatments, and the deviation of the observation (y_{ij}) from its own column mean ($\bar{y}_j$), or within treatments. We can show that this relationship also holds for sums of squared deviations, yielding the partitioned sum of squares.

$$\sum_{j=1}^{c}\sum_{i=1}^{n_j}(y_{ij}-\bar{y})^2 = \sum_{j=1}^{c}\sum_{i=1}^{n_j}\{(\bar{y}_j-\bar{y})+(y_{ij}-\bar{y}_j)\}^2$$

$$=\sum_{j=1}^{c}\sum_{i=1}^{n_j}\{(\bar{y}_j-\bar{y})^2+2(\bar{y}_j-\bar{y})(y_{ij}-\bar{y}_j)+(y_{ij}-\bar{y}_j)^2\}$$

$$=\sum_{j=1}^{c}n_j(\bar{y}_j-\bar{y})^2+\sum_{j=1}^{c}\sum_{i=1}^{n_j}(y_{ij}-\bar{y}_j)^2$$

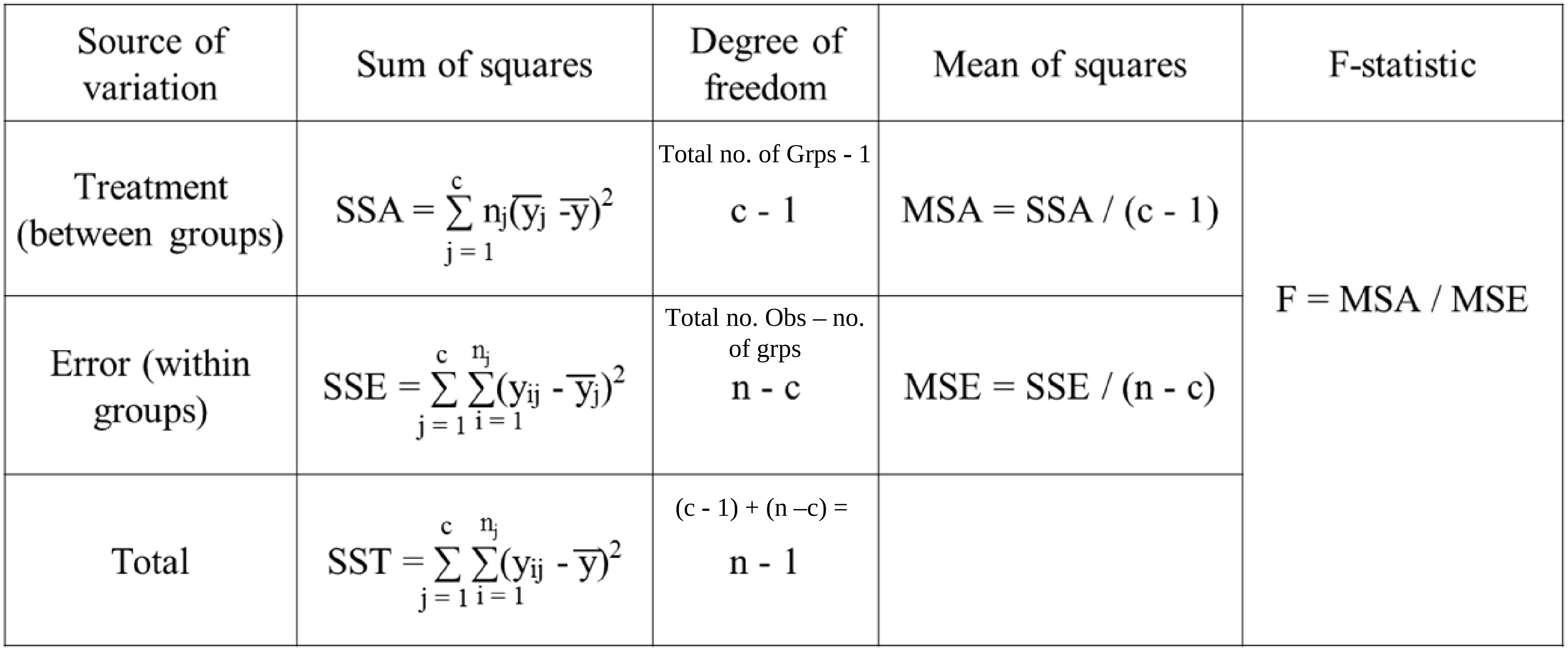
$$\boxed{\sum_{j=1}^{c}\sum_{i=1}^{n_j}(y_{ij}-\bar{y})^2 = \sum_{j=1}^{c}n_j(\bar{y}_j-\bar{y})^2+\sum_{j=1}^{c}\sum_{i=1}^{n_j}(y_{ij}-\bar{y}_j)^2}$$

This important relationship may be expressed simply as

SST = SSA + SSE (Partitioned Sum of Squares)

ie. Sum of squares total = sum of squares between treatments + sum of squares within treatments
(Explained by factor A) (Unexplained random error)

If the treatment means do not differ greatly from grand mean, SSA will be small and SSE will be large (and conversely). The sums SSA and SSE may be used to test the hypothesis that the treatment means differ from the grand mean. However, we first divide each sum of squares by its degree of freedom (to adjust for group sizes). The test statistic is the ratio of the resulting mean squares. These calculations can be arranged in the tabular format as shown in the next slide.



Source of variation	Sum of squares	Degree of freedom	Mean of squares	F-statistic
Treatment (between groups)	$SSA = \sum_{j=1}^{c} n_j(\bar{y}_j - \bar{y})^2$	Total no. of Grps - 1 c - 1	MSA = SSA / (c - 1)	F = MSA / MSE
Error (within groups)	$SSE = \sum_{j=1}^{c}\sum_{i=1}^{n_j}(y_{ij} - \bar{y}_j)^2$	Total no. Obs – no. of grps n - c	MSE = SSE / (n - c)	
Total	$SST = \sum_{j=1}^{c}\sum_{i=1}^{n_j}(y_{ij} - \bar{y})^2$	(c - 1) + (n –c) = n - 1		

The ANOVA calculations are mathematically simple but involve tedious sums. These calculations are almost always done on computer.

Test statistic : At the beginning of this chapter we described the variation in Y as consisting of explained variation and unexplained variation. To test whether the independent variable explains a significant proportion of the variation in Y, we need to compare the explained (due to treatments) and unexplained (due to error) variation. Recall that the F-distribution describes the ration of two variances. Therefore, it makes sense that the ANOVA test statistic is the F-test statistic. The F statistic is the ratio of the variance due to treatments to the variance due to error. MSA is the mean square due to treatment and MSE is the mean square within treatments. Following equation shows the F-statistic and its degree of freedom.

$$F = MSA / MSE = \{(SSA / (c - 1) / SSE / (n - c)\}$$

where, c = number of groups & n = total number of observations

If there is little difference among treatments, we would expect MSA to be near zero because the treatment means y_j would be near the overall mean y. Thus, when F is near zero we would not expect to reject the hypothesis of equal group means. But how large must F be to convince the mean differ? Just as with Z-test or a t-test, we need a decision rule.

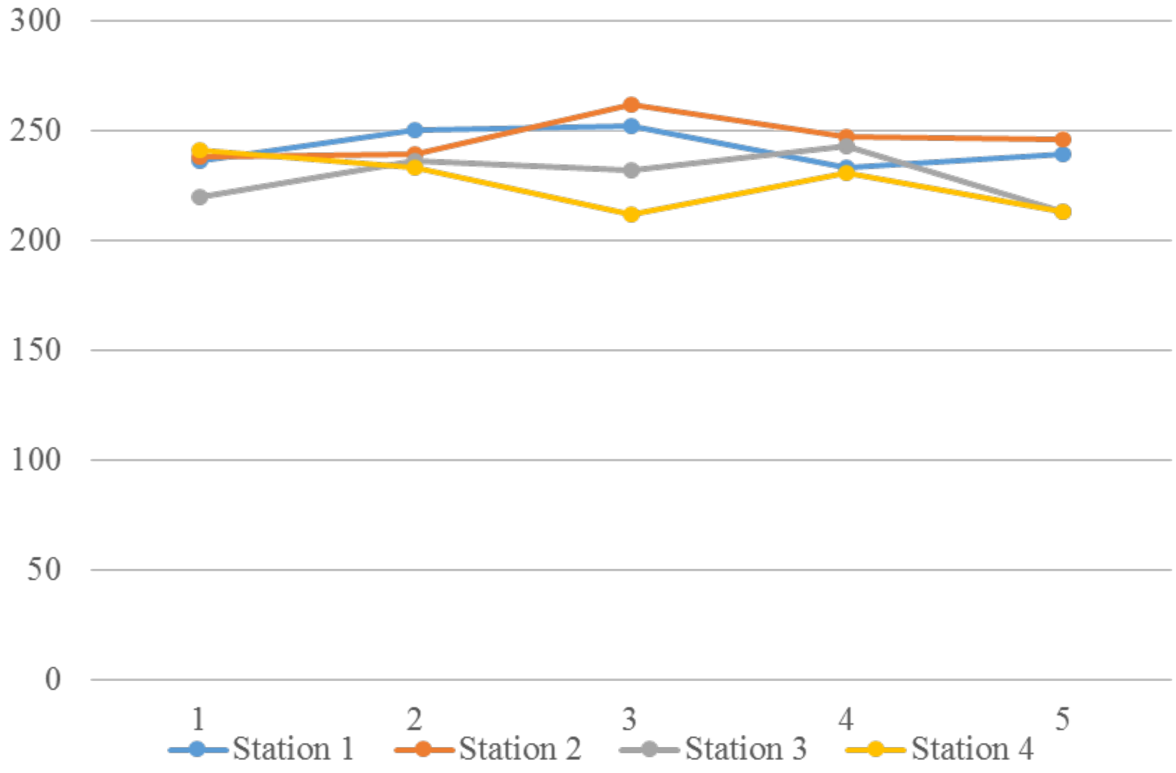
Decision Rule **:** The F-distribution is right-skewed distribution that starts at zero (F can not be negative since variances are sums of squares) and has no upper limit (since the variances can be of any magnitude). For ANOVA the F-test is a right tailed test. For a given level of significance α, we can use F-table to obtain the right-tail critical value of F.

If $F_{calculated} > F_{critical}$ value then the hypothesis is rejected and if $F_{calculated} < F_{critical}$ the hypothesis can not be rejected.

Ex1 : A cosmetic manufacturer’s regional distribution center has four workstations that are responsible for packing cartons for shipment to small retailers. Each workstation is staffed by 2 persons. The task involves assembling each order, placing it in shipping carton, inserting packing material, taping the carton and placing a computer generated shipping label on each carton. Generally, each station can pack 200 cartons a day and often more. However, there is variability, due to differences in orders, labels and cartons. Table below shows the no. of cartons packed per day during a recent week. Is the variation among stations within the range attributable to chance, or do these samples indicate actual differences in the means ?

	Station 1	Station 2	Station 3	Station 4
	236	238	220	241
	250	239	236	233
	252	262	232	212
	233	247	243	231
	239	246	213	213
Sum	1210	1232	1144	1130
Mean	242	246.4	228.8	226
Std deviation	8.515	9.607	12.153	12.884
n	5	5	5	5

Soln : As a preliminary step, we plot the data (as shown below) to check for any time pattern and just to visualize the data. We see some potential differences in means, but no obvious time Pattern (otherwise we would have to consider obsn order as a second factor). We proceed with the hypothesis test.



Step 1 : State the Hypothesis -

$H_0 : \mu_1 = \mu_2 = \mu_3 = \mu_4$ (the means are same)

H_1 : Not all means are equal (at least one mean is different)

Step 2 : State the decision rule -

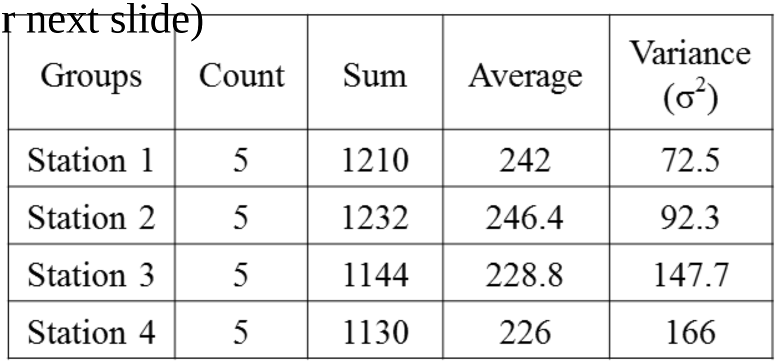
There are c = 4 groups and n = 20 obsns, so degree of freedom for the F-test are –

Numerator : $df_1 = c – 1 = 4 – 1 = 3$ (between treatments, factors)

Denominator : $df_2 = n – c = 20 – 4 = 16$ (within treatments, error)

We will use $\alpha = 0.05$ for the test. The 5% right tail critical value from the F-table is $F_{0.05,3,16} = 3.24$.

Step 3 : Perform the calculations – As per table at significance level 0.05. (Refer next slide)



Groups	Count	Sum	Average	Variance (σ^2)
Station 1	5	1210	242	72.5
Station 2	5	1232	246.4	92.3
Station 3	5	1144	228.8	147.7
Station 4	5	1130	226	166

Groups	Count	Sum	Average	Variance (σ^2)
Station 1	5	1210	242	72.5
Station 2	5	1232	246.4	92.3
Station 3	5	1144	228.8	147.7
Station 4	5	1130	226	166

Degree of freedom –

Between groups = c – 1 = 4 – 1 = 3

Within groups = n – c = 20 – 4 = 16

Total = n – 1 = 20 – 1 = 19

ANOVA Table

Source of variation	Sum of squares	Degree of freedom	Mean square	F_{cal}	P-value	F-critical
Between groups (SSA)	1479.2	3	493.0667	4.121769	0.024124	3.24
Within groups (SSE)	1914	16	119.625			
Total (SST)	3393.2	19				

Step 4 : Make the decision –
Since the test statistic F = 4.12 exceeds the critical value $F_{.05}$ = 3.24, we can reject the hypothesis of equal means. Since excel gives the p-value, you don't actually need excel's critical value. The p-value (0.024124) is less than the the level of significance ($\alpha = 0.05$) which confirms that we should reject the hypo of equal means. The same decision is arrived by Megastat's ANOVA table.

Note : Refer F-table and t-table

Sample calculation

$$\sum_{j=1}^{c}\sum_{i=1}^{n_j}(y_{ij}-\bar{y})^2=\sum_{j=1}^{c}n_j(\bar{y}_j-\bar{y})^2+\sum_{j=1}^{c}\sum_{i=1}^{n_j}(y_{ij}-\bar{y}_j)^2$$

$$\bar{y}_j = 242,\ 246.6,\ 228.8 \text{ and } 226$$

$$Y = (242 + 246.6 + 228.8 + 226) / 4 = 235.8$$

$$SSA = \sum_{j=1}^{4} n_j(\bar{y}_j - \bar{y})^2, \quad df_1 = 4 - 1 = 3$$

$$SSE = \sum_{j=1}^{4}\sum_{i=1}^{5}(y_{ij} - \bar{y}_j)^2, \quad df_2 = 20 - 4 = 16$$

$$SST = \sum_{j=1}^{4}\sum_{i=1}^{5}(y_{ij} - \bar{y})^2$$

$$SSA = 5\{(242 - 235.8)^2 + (246.4 - 235.8)^2 + (228.8 - 235.8)^2 + (226 - 235.8)^2\} = 1479.2$$

Ex 2: ANOVA table for 7 days's mean productivity of a manufacturing house at 4 different production centers is given in the following table –

Source of variation	Sum of squares
Between Production Centers (SSA)	6400
Error (SSE)	8600
Total (SST)	15000

Performing test of hypothesis at a significance level of 5% can we infer that the mean productivity at different production centers are same.

Soln :

Source of variation	Sum of squares	Degree of freedom	Mean square	F_{stat}	$F_{0.05, 3, 24}$
Between Prod Centers (SSA)	6400	4 - 1 = 3	2133.33	5.95	3.01
Error (SSE)	8600	27 – 3 = 24	358.33		
Total (SST)	15000	4 x 7 – 1 = 27			

$F_{critical}$ = 3.01 obtained from F-table. It is evident from the F-distribution (shown below) that $F_{stat} > F_{critical}$. Hence the inference drawn is - the mean productivity at different production centers are not the same.

Experimenter has to initiate corrective action to bring in uniformity in production at all the production centers.

Ex 3: In many integrated circuit manufacturing steps, wafers are completely coated with a layer of material such as silicon dioxide or metal. The unwanted material is then selectively removed by etching through a mask at different levels of watts of radio frequency power. It is interested to test whether the average etching rates of unwanted material at all 4 radio frequency (RF) levels are equal?

RF Power (W)	Observed Etch Rate (A/Min)				
	1	2	3	4	5
160	575	542	530	539	570
180	565	593	590	579	610
200	600	651	610	637	629
220	725	700	715	685	710

Soln : H_0 : The average etching rates of unwanted material at all 4 radio frequency(RF) levels are equal

(vs)

H_1 : The average etching rates at all 4 radio frequency(RF) levels are not equal

RF Power (W)	Observed Etch Rate (A/Min)					Totals y_i	Averages $\bar{y}_{i.}$
	1	2	3	4	5		
160	575	542	530	539	570	2756	551.2
180	565	593	590	579	610	2937	587.4
200	600	651	610	637	629	3127	625.4
220	725	700	715	685	710	3535	707.0
						y_i = 12,355	$\bar{y}_{..}$ = 617.75

$$\text{SST}: \sum_{i=1}^{4} \sum_{j=1}^{5} (y_{ij})^2 - (y_{..})^2 / N$$

$$= (575)^2 + (542)^2 + \ldots.. + (710)^2 - (12{,}355)^2 / 20 = 72{,}209.75$$

$$\text{SSA} = (1/n) \sum_{i=1}^{4} y_{i.}^{\,2} - y_{..}^{\,2}/N = (1/5)[(2756)^2 + \ldots + (3535)^2]$$

$$- (12{,}355)^2/20 = 66{,}870.55$$

We will use the analysis of variance to test

$H_0 : \mu_1 = \mu_2 = \mu_3 = \mu_4$ against the alternative H_1 : Some means are different.

The sums of squares required are computed using Equations 3.8, 3.9 and 3.10 as follows –

$$\text{SSE} = \text{SST} - \text{SSA} = 72{,}209.75 - 66{,}870.55 = 5339.20$$

ANOVA for the Plasma Etching Experiment

Source of variation	Sum of Square	Degree of Freedom	MS	$F_{cal} = (MSA / MSE)$
RF Power	66,870.55	3	MSA = 22,290.18	66.80
Error	5339.20	16	MSE = 333.70	
Total	72,209.75	19		

RF Power or between-treatment mean square (22,290.18) is many times larger than the within-treatment or error mean square (333.70). This indicates that it is unlikely that the treatment means are equal. More formally, we can compute the F ratio $F_{cal} = 22{,}290.18/333.7 = 66.80$ and compare this to an appropriate upper-tail percentage point of the $F_{3,16}$ distribution. To use a fixed significance level approach, suppose that the experimenter has selected $\alpha = 0.05$. From F-table we find that $F_{0.05,3,16} = 3.24$. Because F_{cal} (66.80) > $F_{critical}$ (3.24), we reject Ho and conclude the treatment means differ; that is, the RF Power setting significantly affects the mean etch rate.

Ex 4: Suppose 3 drying formulas for curing a glue are studied and the following times (in minutes) are observed –

Formula A	Formula B	Formula C
13	13	4
10	11	1
8	14	3
11	14	4
8		2
		4

Test whether mean curing times due to 3 different formulae are equal at 0.01 level of significance.

Soln :

H_0 : Mean curing times due to 3 different formulae are equal i.e., $\mu_1 = \mu_2 = \mu_3$

(vs)

H_1 : Mean curing times due to 3 different formulae are not equal

Curing Times of A	Curing Times of B	Curing Times of C	y_1^2	y_2^2	y_3^2
y_1	y_2	y_3			
13	13	4	169	169	16
10	11	1	100	121	1
8	14	3	64	196	9
11	14	4	121	196	16
8		2	64		4
		4			16
Total 50	52	18	518	682	62

$K = 3,\ n_1=5,\ n_2 = 4,\ n_3 = 6,\ n = 15,\ T = 50 + 52 + 18 = 120,$

$T(y^2) = 518 + 682 + 62 = 1262,\ SST = 1262 - (120^2 / 15) = 302,$

$SSA = (50^2/5) + (52^2/4) + (18^2/6) - (120^2/15) = 270,\ SSE = 302 - 270 = 32$

Anova Table

Source of variation	Sum of squares	Degree of freedom	Mean squares	F_{cal} (MSA/MSE)	$F_{critical}$ $[F_{\alpha,(k-1),(n-k}]$
Treatments (Formulae)	SSA = 270	k - 1 = 3-1 = 2	MSA = SSA/(k – 1) = 270/2 = 135	135/2.667 = 50.6	$F_{0.01,(2, 12)}$ = 6.93
Error	SSE = 32	n - k = 15 – 3 = 12	MSE = SSE/(n – k) = 32/12 = 2.667		
Total	SST = 302	n - 1 = 15 – 1 = 14			

Conclusion : Since F_{cal} 50.6 > $F_{critical}$ (6.93), H_0 is rejected ie. Mean curing times due to 3 different formulae are not equal.

Ex 5: I belong to a golf club in my neighbourhood. I divide the year into three seasons – Summer (June – September), Winter (November – March) and Shoulder (October, April & May). I believe that I play my best golf during Summer (because I have more time and the course isn't crowded and Shoulder (because the course isn't crowded) season and my worst golf is during the Winter (because when all of the part-year residents show up, the course is crowded, play is slow and I get frustrated). Data from the last year are shown in the following table.

Season	Observations									
Summer	83	85	85	87	90	88	88	84	91	90
Shoulder	91	87	84	87	85	84	83			
Winter	94	91	87	85	87	91	92	86		

Do the data indicate that my opinion is correct? Use α = 0.05.

Soln : H_0 : The average golf scores are equal between among three seasons

(vs)

H_1: The average scores are not equal

From each observation 85 is subtracted then we obtain

Summer : -2, 0, 0, 2, 5, 3, 3, -1, 6, 5; $T_1=21$, $T_1^2 = 441$, $T_1^2/10 = 44.1$

Shoulder : 6, 2, -1, 2, 0, -1, -2; $T_2 = 6$, $T_2^2 /7 = 36/7 = 5.14$

Winter : 9, 6, 2, 0, 2, 6, 7, 1; $T_3 = 33$, $T_3^2/8 = 136.125$

$k = 3$, $n = 10 + 7 + 8 = 25$, $T = 21 + 6 + 33 = 60$, $T^2/n = 60^2/25 = 144$

$T(y^2) = (-2)^2 + (0)^2 + \dots + (7)^2 + (1)^2 = 374$

$SST = T(y^2) - T^2/n = 374 - 144 = 230$

$SSA = T(T_i^2/n_i) - T^2/n = 44.1 + 5.14 + 136.125 - 144 = 41.365$

$SSE = SST - SSA = 230 - 41.365 = 188.635$

$MSA = SSA/(k-1) = 41.365/2 = 20.6825,$

$MSE = SSE/(n-k) = 188.635/22 = 8.5743$

$F_{cal} = MSA / MSE = 20.365/8.5743 = 2.375$

$F_{((k-1) = 2,(n-k) = 22)}$ at 0.05 level of significance is 3.44
ie. $F_{0.05, 2, 22} = 3.44$

Since $F_{cal} < F_{critical}$ then we accept H_0.

Source of variation	SS	DF	MS	F_{cal}	$F_{critical}$
Treatment (Between Seasons)	SSA = 41.365	k - 1 = 2	MSA = 20.6825	MSA/MSE = 2.412	3.44
Error	SSE = 188.635	n - k = 22	MSE = 8.5743		
Total	SST = 230	n - 1 = 24			

Conclusion : Since F_{cal} (2.412) < $F_{critical}$ (3.44), accept the Null Hypothesis ie. golf scores are equal in all the sthree seasons.

Practice Problems

Q1. Highlight the difference between one-way and two-way Anova.

Q2. Conduct test of hypothesis at a significance level of 5% to infer whether mean productivity at different work stations are same. Analysis of variance for the Average productivity for 7 days at 5 different workstations is tabulated below.

Source of variation	Sum of squares
Between Production Centers (SSA)	4650
Error (SSE)	13730

Q3. Analysis of Variance for the mean defects of six days' production at four different production centers is given in the following table.

Source of variation	Sum of squares
Between Production Centers (SSA)	2800
Error (SSE)	3900

Practice Problems

Q4. Three different kinds of food are tested on three groups of rats for 5 weeks. The objective is to check the difference in mean weight (in grams) of the rats per week. Apply one-way ANOVA using a 0.05 significance level to the following data –

Food I	Food II	Food III
8	4	11
12	5	8
19	4	7
8	6	13
6	9	7
11	7	9

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

BSDCHZC355
Statistical Inference & Applications

Shaibal Kr. Sen
Session 05

innovate achieve lead

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

STATISTICAL INFERENCES

&

APPLICATIONS

Scheduled Topics to be covered

Two –way ANOVA Classification

A two-way ANOVA tests the effect of two independent variables on a dependent variable on the expected outcome along with their relationship to the outcome itself. Random factors would be considered to have no statistical influence on a data set, while systematic factors would be considered to have statistical significance. Whereas a one-way ANOVA is a type of statistical test that compares the variance in the group means within a sample whilst considering only one independent variable or factor. It is a hypothesis-based test, meaning that it aims to evaluate multiple mutually exclusive theories about our data.

By using ANOVA, a researcher is able to determine whether the variability of the outcomes is due to chance or to the factors in the analysis. ANOVA has many applications in Finance, Economics, Science, Engineering, DOE, Medicine, and Social Science.

An ANOVA test is the first step in identifying factors that influence a given outcome. Once an ANOVA test is performed, a tester may be able to perform further analysis on the systematic factors that are statistically contributing to the data set’s variability.

Analysis of variances is helpful for testing the effects of variables on one another. It is similar to multiple two-sample t-tests. However, it results in fewer type 1 errors and is appropriate for a range of issues. An ANOVA test groups differences by comparing the means of each group and includes spreading out the variance across diverse sources. It is employed with subjects, test groups, between groups and within groups.

One-Way ANOVA -vs- Two-Way ANOVA **:** There are two main types of analysis of variance - one-way (or unidirectional) and two-way (bidirectional). One-way or two-way refers to the number of independent variables in your analysis of variance test. A one-way ANOVA evaluates the impact of a sole factor on a sole response variable. It determines whether the observed differences between the means of independent (unrelated) groups are explainable by chance alone, or whether there are any statistically significant differences between groups.

A two-way ANOVA is an extension of the one-way ANOVA. With a one-way, you have one independent variable affecting a dependent variable. With a two-way ANOVA, there are two independents. For example, a two-way ANOVA allows a company to compare worker productivity based on two independent variables, such as department and gender. It is utilized to observe the interaction between the two factors. It tests the effect of two factors at the same time.

A three-way ANOVA, also known as three-factor ANOVA, is a statistical means of determining the effect of three factors on an outcome.

The only difference between one-way and two-way ANOVA is the number of independent variables. A one-way ANOVA has one independent variable, while a two-way ANOVA has two. Examples of -

One-way ANOVA - Testing the relationship between shoe brand (Nike, Adidas, Saucony, Hoka) and race finish times in a marathon.

Two-way ANOVA - Testing the relationship between shoe brand (Nike, Adidas, Saucony, Hoka), runner age group (junior, senior, master’s), and race finishing times in a marathon.

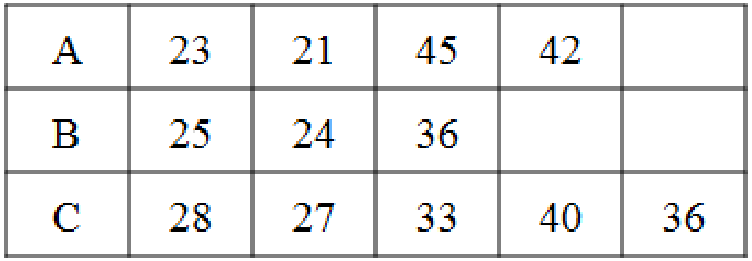
All ANOVAs are designed to test for differences among three or more groups. If you are only testing for a difference between two groups, use a t-test instead.

As we have seen earlier Two-way ia an extension of the one way ANOVA in terms of taking the second factor into the analysis.

One-way or two-way refers vto the number of independent variables in the experiment.

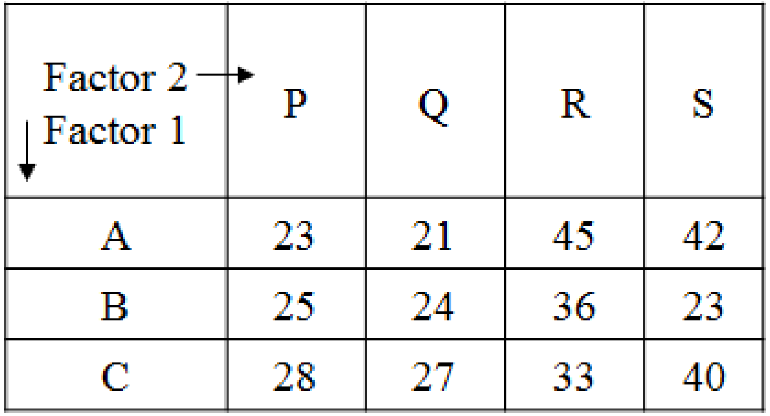
It is important to note that two-way ANOVA test is used when the number of observations in the subclasses are equal.

One-way ANOVA



A	23	21	45	42	
B	25	24	36		
C	28	27	33	40	36

Two-way ANOVA



Factor 2 → Factor 1 ↓	P	Q	R	S
A	23	21	45	42
B	25	24	36	23
C	28	27	33	40

Sum of the Squares (SS) -

Correction Factor (CF) = G^2/N; where, G is grand total, N is total number of elements.

(1)SS between Rows = $\Sigma\{(T_i)^2/n_i\} - CF$; where, T_i is total of ith row.

(2)SS between columns = $\Sigma\{(T_j)^2/n_j\} - CF$; where, T_j is total of jth column.

(3)SS Total = $\Sigma\Sigma(x_{ij})^2 - CF$

(4)Within Error = (3) – (1) – (2)

Ex 1 **:** Three varieties of coal were analysed by four chemists and the ash contents in the varieties were found as tabulated below -

Varieties	Chemists			
	I	II	III	IV
A	8	5	5	7
B	7	6	4	4
C	3	6	5	4

Discuss the significance of the difference - (i) Chemists and (ii) Varieties of coal in respect of Ash Content.
Test at 5% levcl of significance

Soln : For Varieties - H_{0V} **:** μ_V ie. $\mu_A = \mu_B = \mu_C = \mu$

H_{1V} **:** $\mu_V \neq \mu$

For Chemists - H_{0C} **:** μ_C ie. $\mu_I = \mu_{II} = \mu_{III} = \mu_{IV} = \mu$

H_{1C} **:** $\mu_C \neq \mu$

Soln :

Source of Variation	DF	SS	MSS = SS/DF	F-ratio
Between Varieties	3 - 1 = 2			
Between Chemists	4 - 1 = 3			
Within Error	(3-1)(4 - 1) = 6			
Total	3x4 – 1 = 11			

Varieties	Chemists				Total
	I	II	III	IV	
A	8	5	5	7	25
B	7	6	4	4	21
C	3	6	5	4	18
Total	18	17	14	15	G =64

Correction Factor (CF) = $G^2/N = 64^2/3x4 = 4096/12 = 341.33$

SS Total = $\Sigma\Sigma(x_{ij})^2$ – CF = $8^2 + 5^2 + \ldots + 5^2 + 4^2 - 341.33 \approx 24.667$

SS between Varieties = $\Sigma\{(T_i)^2/n_i\}$ – CF = $(25^2 + 21^2 + 18^2)/4 - 341.33$

≈ 6.167

SS between Chemists = $\Sigma\ \{(T_j)^2/n_j\}$ – CF

$= (18^2 + 17^2 + 14^2 + 15^2)/3 - 341.33 \approx 3.333$

SSError = SS Total – SS between Varieties – SS between Chemists

$= 24.667 - 6.167 - 3.333 = 15.167$

ANOVA Table

Source of Variation	DF	SS	MSS = SS/DF	$F_{calculated}$	$F_{critical}$ (From F-table)
Between Varieties	3 - 1 = 2	6.167	6.167/2 = 3.083	MSV/MSE = 1.219	$F_{0.05,2,6} = 5.14$
Between Chemists	4 - 1 = 3	3.333	3.333/3 = 1.111	MSC/MSE = 0.439	$F_{0.05,3,6} = 4.76$
Within Error	(3-1)(4 - 1) = 6	15.167	15.167/6 = 2.528		
Total	3x4 – 1 = 11	24.667			

Since F_{cal} (1.219) < $F_{critical}$ (5.14) for Varieties we can not reject the null hypothesis ie. there is no significant difference between the Varieties.

Since F_{cal} (0.439) < $F_{critical}$ (4.76) for Chemists we can not reject the null hypothesis ie. there is no significant difference between the Chemists.

Ex 2 : Complete the following Anova Table assuming α = 0.05 -

Source of Variation	DF	SS	MSS = SS/DF	$F_{calculated}$	$F_{critical}$ (From F-table)
Between Blocks	4	26.8			
Between Treatments	3		2.5		
Within Error					
Total		85.3			

Soln : From the information given in the table, we find there are two variables viz. – Blocks & Treatments. Hence, it is a two-way ANOVA with Number of Blocks = DF + 1 = 4 + 1 = 5 and Number of Blocks = DF + 1 = 3 + 1 = 4.

We know in a two-way ANOVA number of elements in the blocks (rows) & treatments (columns) are same ie. 5 & 4 respectively. Hence, total DF = 5x4 – 1 =19.

Source of Variation	DF	SS	MSS = SS/DF	$F_{calculated}$	$F_{critical}$ (From F-table)
Between Blocks	4	26.8	26.8/4 = 6.7	6.7/2.5 = 2.68	$F_{0.05,\ 4,\ 12} = 3.26$
Between Treatments	3	28.5	28.5/3 = 9.5	9.5/2.5 = 3.8	$F_{0.05,\ 3,\ 12} = 3.49$
Within Error	19 – (4 + 3) = 12	x = 30	2.5		
Total	5x4 – 1 = 19	85.3			

Soln **:** From the information given in the table, we find there are two variables viz. – Blocks & Treatments. Hence, it is a two-way ANOVA with Number of Blocks = DF + 1 = 4 + 1 = 5 and Number of Blocks = DF + 1 = 3 + 1 = 4.

We know in a two-way ANOVA number of elements in the blocks (rows) & treatments (columns) are same ie. 5 & 4 respectively. Hence, total DF = 5x4 – 1 =19.

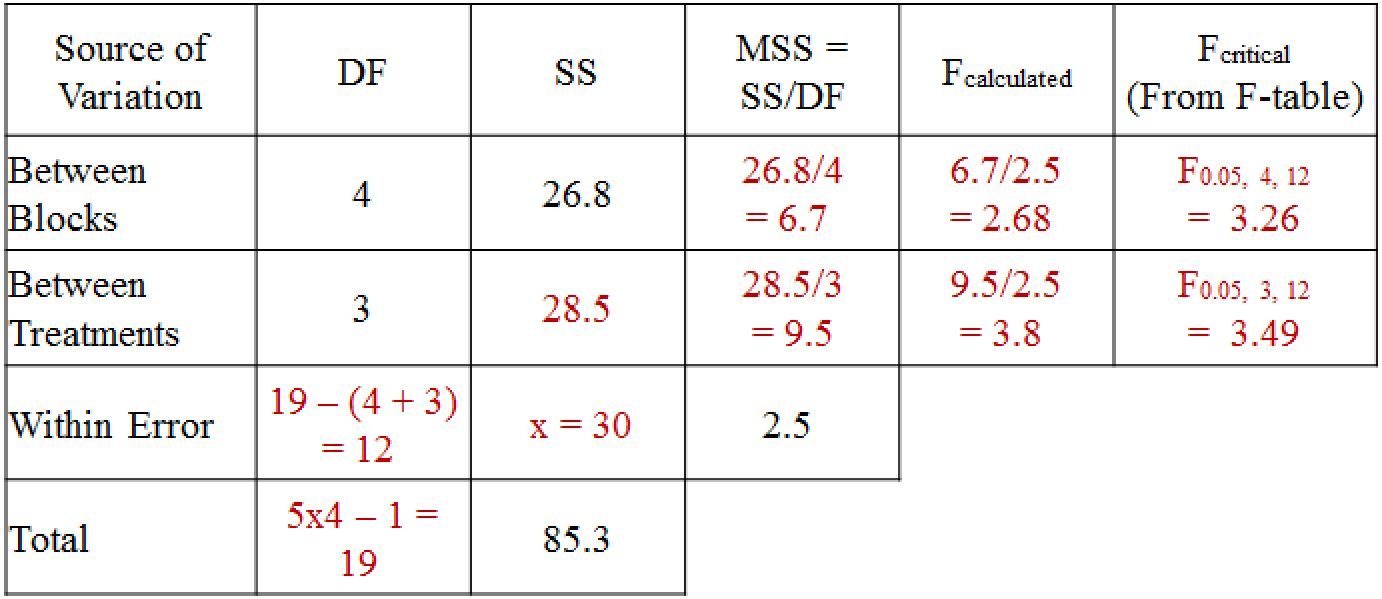
So, DF of Treatments = 19 – (4 + 3) = 12.

Source of Variation	DF	SS	MSS = SS/DF	$F_{calculated}$	
Between Blocks	5	32.7	32.7/5 = 6.54		
Between Treatments	4	85.3 – (32.7 + 20) = 32.6	32.6/4 = 8.15		
Within Error	29 – (5 + 4) = 20	20x1 = 20	1		
Total	6x5 – 1 = 29	85.3			

Let SS Treatments be x. As per given data x/12 = 2.5 ie. x = 30

Since, SS Total = SS Blocks + SS Treatments + SS Error

or, 85.3 = 26.8 + SS Treatments + 30 Hence, SS Treatments = 85.3 – 56.8 = 28.5



Source of Variation	DF	SS	MSS = SS/DF	$F_{calculated}$	$F_{critical}$ (From F-table)
Between Blocks	4	26.8	26.8/4 = 6.7	6.7/2.5 = 2.68	$F_{0.05,\ 4,\ 12}$ = 3.26
Between Treatments	3	28.5	28.5/3 = 9.5	9.5/2.5 = 3.8	$F_{0.05,\ 3,\ 12}$ = 3.49
Within Error	19 – (4 + 3) = 12	x = 30	2.5		
Total	5x4 – 1 = 19	85.3			

Since, for blocks F_{cal} (2.68) < $F_{critical}$ (3.26) null hypothesis can not be rejected but for treatments F_{cal} (3. 8) > $F_{critical}$ (3.49) null hypothesis can not be accepted.

Ex 3 **:** A medical device manufacturer produces vascular grafts (artificial veins). These grafts are produced by extruding billets of polytetrafluoroethylene (PTFE) resins combined with a lubricant into tubes. Frequently, some of the tubes in a production run contain small, hard protrusions on the external surface. These defects are known as “flicks”. The defect is cause for rejection of the unit.

The product developer responsible for the vascular grafts suspects that the extrusion pressure affects the occurrence of flicks and therefore intends to conduct an experiment to investigate this hypothesis. However, the resin is manufactured by an external supplier and is delivered to the medical device manufacturer in batches. The engineer also suspects that there may be significant batch-to-batch variation.

Please analyse and infer your finding at 5% LOS.

		F_2					
	Extrusion Pressure (PSI)	Batch of Resin (Block)					
		1	2	3	4	5	6
F_1	8500	90.3	89.2	98.2	93.9	87.4	97.9
	8700	92.5	89.5	90.6	94.7	87.0	95.8
	8900	85.5	90.8	89.6	86.2	88.0	93.4
	9100	82.5	89.5	85.6	87.4	78.9	90.7

Soln :

		F_2						
	Extrusion Pressure (PSI)	Batch of Resin (Block) [q or b = 6]						Treatment Total
		1	2	3	4	5	6	
F_1 [p or a = 4]	8500	90.3	89.2	98.2	93.9	87.4	97.9	556.9
	8700	92.5	89.5	90.6	94.7	87	95.8	550.1
	8900	85.5	90.8	89.6	86.2	88	93.4	533.5
	9100	82.5	89.5	85.6	87.4	78.9	90.7	514.6
Block Totals		350.8	359	364	362.2	341.3	377.8	y.. = 2155.1

Null Hypothesis H_0F_1 : All extrusion pressures equally affect the mean yield.

H_0F_2 : All batches of resins are equally effective.

To perform analysis of variance, we need the following sums of squares –

SSTotal = $\sum_{i=1}^{4}\sum_{j=1}^{6}(y_{ij})^2 - (y_{..})^2/N = 193{,}999.31 - (2155.1)^2/24 = 480.31$

SSTreatment = $(1/b)\{\sum_{i=1}^{4}(y_{i.})^2\} - (y_{..})^2/N$

$= (1/6)\{(556.9)^2 + (550.1)^2 + (533.5)^2 + (514.6)^2\} - (2155.1)^2/24 = 178.17$

SSBlocks = $(1/a)\{\sum_{j=1}^{6}(y_{.j})^2\} - (y_{..})^2/N$

$= (1/4)\{(350.8)^2 + (359.0)^2 + \ldots + (377.8)^2\} - (2155.1)^2/24 = 192.25$

SSE = SSTotal – SSTreatment – SSBlocks = 480.31 – 178.17 – 192.25 = 109.89

The ANOVA table and inferences are shown in the next slide.

ANOVA Table for Vascular Graft Experiment

Sources of Variation	Sunm of Squares (SS)	Degrees of Freedom (DF)	Mean Square (SS/DF)	F_{cal}	P-value
Treatments (Extrusion Pressure)	178.17	4 – 1 = 3	178.17/3 = 59.39	59.39/7.33 = 8.10	0.0019 < 0.05
Blocks (Batches)	192.25	6 – 1 = 5	192.25/5 = 38.45	38.45/7.33 = 5.24	0.005 < 0.05
Error	109.89	15	109.89/15 = 7.33		
Total	480.31	4x6 – 1 = 23			

Conclusion **:** Using $\alpha = 0.05$, the critical value of F ie. $F_{0.05,3,15} = 3.29$ & $F_{0.05,5,15} = 2.9$. Since the calculated values of F for both extrusion pressure (8.10) and resin batch effect (5.24) are beyond the acceptance region $F_{critical}$ values (3.29 & 2.9), we conclude that null hypothesis can not be accepted ie. extrusion pressure affects the mean yield and resin batches seem to differ significantly. It is also evident from the small P-value ($< \alpha = 0.05$).

Ex 4 :

Solve using Two-way ANOVA method

Observation	A	B	C
1	1,4,0,7	13,5,7,15	9,16,18,13
2	15,6,10,13	6,18,9,15	14,7,6,13

Soln :

Row and column sums

	A	B	C	Row total (x_a)
1	12	40	56	108
2	44	48	40	132
Col total (x_b)	56	88	96	240

$\sum x^2 = 1^2 + 4^2 + 0^2 + \ldots + 7^2 + 6^2 + 13^2 = 3010$ …. (A)

$\sum (x_b)^2/ra = (56^2 + 88^2 + 96^2)/(4 \times 2) = (3136 + 7744 + 9216)/8 = 20096/8 = 2512$…. (B)

$\sum(x_a)^2 /rb = (108^2 + 132^2)/4x3 = (11664 + 17424)/12 = 29088/12 = 2424$ …. (C)

$\sum\sum(x_{ab})^2/r = (12^2 + 40^2 + 56^2 + 44^2 + 48^2 + 40^2)/4$

$= (144 + 1600 + 3136 + 1936 + 2304 + 1600)/4 = 10720/4 = 2680$

$(\sum x)^2/rab = (240)^2/(4x2x3) = 57600/24 = 2400$….(D)

Sum of squares total

$SST = \sum x^2 - (\sum x)^2/n = (A) - (D) = 3010 – 2400 = 610$

Sum of squares between rows

$SSA = \sum(x_a)^2/rb - (\sum x)^2/rab = (C) - (D) = 2424 – 2400 = 24$

Sum of squares between columns

$SSB = \sum(x_b)^2/ra - (\sum x)^2/n = (B) - (D) = 2512 – 2400 = 112$

Sum of squares between columns

$SSAB = \sum\sum(x_{ab})^2/r - (\sum x)^2/rab – SSA – SSB = 2680 – 2400 – 24 - 112 = 144$

= 2680 – 2400 – 24 – 112 = 144

Sum of squares Error (residual)

SSE = SST – SSA – SSB – SSAB = 610 – 24 – 112 – 144 = 330

ANOVA Table

Source of Variation	Sum of Squares (SS)	Degree of freedom (DF)	Mean Square (MS)	F_{cal}	P-value
A	SSA = 24	a - 1 = 1	MSR = 24/1 = 24	24/18.333 = 1.3091	0.2675
B	SSB = 112	b - 1 = 2	MSC = 112/2 = 56	56/18.333 = 3.0545	0.0721
AB	SSAB = 144	(a - 1)(b - 1) = 2	MSAB = 144/2 = 72	72/18.333 = 3.9273	0.0384
Error (Residual)	SSE = 330	rab - ab = 18	MSE = 330/18 = 18.333		
Total	SST = 610	rab - 1 = 23			

Conclusion :

1. F for between columns for df 1,2 and LOS $\alpha = 0.05$

ie. $F_{0.05,1,2} = 4.4139$

As calculated F_{cal} (between rows) = $1.3091 < 4.4139$

So, H_0 is accepted, Hence there is no significant difference between rows.

2. F for between columns for df 2, 2 and LOS $\alpha = 0.05$

$F_{0.05,2,2} = 3.5546$

As calculated F_{Cal} (between columns) = $3.0545 < 3.5546$

So, H_0 is accepted, Hence there is no significant difference between columns.

Ex 5 **:** Solve using Two-way ANOVA method

Observation	A	B	C	D	E	F
1	1200	1000	980	900	750	800
2	1000	1100	700	800	500	700
3	890	650	1100	900	400	350

Soln :

	A	B	C	D	E	F	Row total (x_r)
1	1200	1000	980	900	750	800	5630
2	1000	1100	700	800	500	700	4800
3	890	650	1100	900	400	350	4290
Col total (x_c)	3090	2750	2780	2600	1650	1850	14720

$\sum x^2 = 13010000$…. (A)

$\sum(x_c)^2/\ r = (3090^2 + 2750^2 + 2780^2 + 2600^2 + 1650^2 + 1850^2)/3$

$= (9548100 + 7562500 + 7728400 + 6760000 + 2722500 + 3422500)/3 = 937744000/3$

$= 12581333.3333$…. (B)

$\sum(x_r)^2/c = (5630^2 + 4800^2 + 4290^2)/6 = (31696900 + 23040000 + 18404100)/6$

$= 73141000/6 = 12190166.67$…. (C)

$= (\sum x)^2/n = (14720)^2/18 = 216678400/18 = 12037688.8889$…. (D)

Sum of squares total

$SST = \sum x^2 - (\sum x)^2/n = (A) - (D) = 13010000 - 12037688.89 = 972311.11$

Sum of squares between rows

$SSR = \sum(x_r)^2/c - (\sum x)^2/n = (C) - (D) = 12190166.67 - 12037688.89 = 152477.78$

Sum of squares between columns

SSC = $\sum(x_c)^2/r - (\sum x)^2/n$ = (B) - (D) =12581333.33 - 12037688.89 = 543644.44

Sum of squares Error (residual)

SSE = SST – SSR – SSC = 972311.11 - 152477.78 - 543644.44 = 276188.89

ANOVA table

Source of Variation	Sums of Squares (SS)	Degrees of freedom (DF)	Mean Squares (MS)	F_{cal}
Between rows	SSR = 152477.78	r - 1 = 2	MSR = 152477.78/2 = 76238.89	76238.89/27618.89 = 2.7604
Between columns	SSC = 543644.44	c - 1 = 5	MSC = 543644.44/5 = 108728.88	108728.89/27618.89 = 3.9368
Error (residual)	SSE = 276188.89	(r - 1)(c - 1) = 10	MSE = 276188.89/10 = 27618.89	
Total	SST = 972311.11	rc - 1 = 17		

Example of One way ANOVA

Ex 6 : ANOVA table for 7 days's mean productivity of a manufacturing house at 4 different production centers (pc) is given in the following table –

Source of variation	Sum of squares
Between Production Centers (SSA)	6400
Error (SSE)	8600
Total (SST)	15000

Performing test of hypothesis at a significance level of 5% can we infer that the mean productivity at different production centers are same.

Soln : H_0 : Mean Productivity of all the four Centers are the same.

H_1 : Mean Productivity of all the four Centers are not the same.

Source of variation	Sum of squares	Degree of freedom	Mean square	F_{stat}	$F_{0.05, 3, 24}$
Between Prod Centers (SSA)	6400	c – 1 = 4 - 1 = 3	6400/3 = 2133.33	MSA/MSE = 5.95	3.01
Error (SSE)	8600	n - c = 4x7 – 4 = 24	8600/24 = 358.33		
Total (SST)	15000	4 x 7 – 1 = 27			

$F_{critical}$ = 3.01 obtained from F-table. It is evident from the F-distribution (shown below) that $F_{stat} > F_{critical}$. Hence the inference drawn is - the mean productivity at different production centers are not the same.

Experimenter has to initiate corrective action to bring in uniformity in production at all the production centers.

Practice Problems

Q1. A study examining differences in life satisfaction between young adult, middle adult, and older adult men and women was conducted. Each individual who participated in the study completed a life satisfaction questionnaire. A high score on the test indicates a higher level of life satisfaction. Test scores are recorded below.

Group	Young Adult	Middle Adult	Older Adult
Male	4	7	10
	2	5	7
	3	7	9
	4	5	8
	2	6	11
Female	7	8	10
	4	10	9
	3	7	12
	6	7	11
	5	8	13

contd.

(i) Complete the following ANOVA table.

Source	SS	df	MS	F
Age	180	2		
Gender	30	1		
Age x Gender	0	2		
Within	44	24		
Total				

(ii) Are there any significant main effects or an interaction effect.

(iii) Interpret your answer.

Q2. Perform a 2-way ANOVA on the data given below -

Teachers	Students				
	I	II	III	IV	V
A	30	24	33	36	27
B	26	29	24	31	35
C	38	28	35	30	35

(i) Shift the origin to 30. Perform the ANOVA for the transformed data.

(ii) How do the results compare with those obtained for the original data?

Q3. A research study was conducted to examine the impact of eating a high protein breakfast on adolescents' performance during a physical education physical fitness test. Half of the subjects received a high protein breakfast and half were given a low protein breakfast. All of the adolescents, both male and female, were given a fitness test with high scores representing better performance. Test scores are recorded below.

Group	High Protein	Low Protein
Males	10	5
	7	4
	9	7
	6	4
	8	5
Females	5	3
	4	4
	6	5
	3	1
	2	2

contd.

(i) Complete the following table -

Source	SS	df	MS	F
Protein Level	20	1		
Gender	45	1		
Protein Level x Gender	5	1		
Within	36	16		
Total				

(ii) Are there any significant main effects or an interaction effect.

(iii) Interpret your answer.

Q4. Three different kinds of food are tested on three groups of rats for 5 weeks. The objective is to check the difference in mean weight (in grams) of the rats per week. Apply one-way ANOVA using a 0.05 significance level to the following data -

Food I	Food II	Food III
8	4	11
12	5	8
19	4	7
8	6	13
6	9	7
11	7	9

Q5. On the basis of following information, complete the ANOVA Table and solve using Two-way ANOVA method

Observation	A	B	C
1	10, 8, 7, 9, 6	7, 4, 3, 2	11, 9, 10, 9, 11
2	1, 2, 1, 4, 2	6, 7, 6, 5	4, 3, 6, 4, 3
3	3, 2, 3, 3, 4	2, 1, 2, 3	5, 6, 4, 5, 5

ANOVA Table

Source of Variation	Sums of Squares (SS)	Degrees of freedom (DF)	Mean Squares (MS)	F
A	SSA = 145.33			
B	SSB = 45.24			
AB	SSAB = 93.33			
Error (residual)	SSE = 48			
Total				

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

BSDCHZC355
Statistical Inference & Applications

Shaibal Kr. Sen
Session 06

innovate achieve lead

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

STATISTICAL INFERENCES

&

APPLICATIONS

Scheduled Topics to be covered

Design of Experiments : Complete Block Designs

Completely Randomized Design (CRD)

Randomized Block Design (RBD)

Completely Randomized Design (CRD)

The simplest design using the two essential principles of replication and randomization is the CRD. Suppose that we have t treatments (or t levels of a factor) under comparison and the i^{th} treatment is to be replicated r_i times, for $i = 1, 2, \ldots, t$. Then the total number of experimental units necessary for this experiment is $n = \sum_{i=1}^{t} r_i$. In the CRD, we allocate the t treatments completely at random to the n units subject to the condition that the i^{th} treatment appears in r_i units, for $i = 1, 2, \ldots, t$. A Particular case of this equal replication for different treatments, where $r_1 = r_2 = \ldots = r_t = r$, so that $n = rt$.

Description of Design -

Simplest design to use.
Design can be used when experimental units are essentially homogeneous.
Because of the homogeneity requirement, it may be difficult to use this design for field experiments.
The CRD is best suited for experiments with a small number of treatments.

Randomization Procedure -

Treatments are assigned to experimental units completely at random.
Every experimental unit has the same probability of receiving any treatment.
Randomization is performed using a random number table, computer programme etc.

Example of Randomization **:** Given that there are 4 treatments (A, B, C and D) and there are 5 replicates, how many experimental units would you have?

1 D	2 D	3 B	4 C	5 D	6 C	7 A	8 A	9 B	10 D
11 C	12 B	13 A	14 B	15 C	16 B	17 C	18 D	19 A	20 A

Note that there is no "blocking" of experimental units into replicates.

Every experimental unit has the same probability of receiving any treatment.

Advantages of a CRD –

1. Very flexible design (ie. number of treatments & replications is only limited by the available number of experimental units).
2. Statistical analysis is simple compared to other designs.
3. Loss of information due to missing data is small compared to other designs due to the larger number of degrees of freedom for the error source of variation.

Disadvantages –

1. If experimental units are not homogeneous and you fail to minimize this variation using blocking, there may be a loss of precision.
2. Usually the least efficient design unless experimental units are homogeneous.
3. Not suited for large number of treatments.

The analysis of CRD is exactly same as that of One-Way ANOVA

Linear Additive Model for CRD -

$$Y_{ij} = \mu + \tau_i + e_{ij} \quad \text{where,}$$

Y_{ij} is the j^{th} observation of i^{th} treatment,

μ is the population mean,

τ_i is the treatment effect of i^{th} treatment and

e_{ij} is random error with mean 0 and variance σ^2.

After analysis it becomes –

SS Total = SS due to treatment + SS due to error

$$(S_{total})^2 = (S_{Treatment})^2 + (S_E)^2$$

We are interested in testing $H_0 : \tau_1 = \tau_2 = \dots \tau_t$ against the alternatives that τ's are not all equal. The analysis in the present case is the same as that of one-way classified data learnt earlier (in Session 4 & 5).

ANOVA Table for CRD

Sources of Variation	Sum of Square (SS)	Degree of Freedom (d.f.)	Mean Sum of Square (MSS)	Variation Ratio (F_{cal})
Between Treatments	$(S_{Treatment})^2$	t - 1	$MS_{Treatment} = MS_T$ $= (S_{Treatment})^2/(t - 1)$	MS_T/MS_E
Error	$(S_{Error})^2$	n - t	$MS_{Error} = MS_E$ $= (S_{Error})^2/(n - 1)$	
Total	$(S_{Total})^2$	n - 1		

We reject H_0 at the LOS α if F_{cal} $[MS_T/MS_E] > F_{critical}$ $[F_{\alpha, (t-1), (n-t)}]$; otherwise H_0 is accepted. When H_0 is rejected, we may decide further course of action.

Ex 1 : Given the following data, test whether mean effects due to the treatments are equal.

Treatment

A	B	C	D
2.0	1.7	2.0	2.1
2.2	1.9	2.4	2.2
1.8	1.5	2.7	2.2
2.3		2.5	1.9
1.7		2.4	

Soln : Step 1

Hypothesis to be tested $H_0 : \mu_1 = \mu_2 = \mu_3 = \mu_4$ - vs -

Alternative Hypothesis H_1 : At least one of the means is different

	Treatment			
Replicate	A	B	C	D
1	2.0	1.7	2.0	2.1
2	2.2	1.9	2.4	2.2
3	1.8	1.5	2.7	2.2
4	2.3		2.5	1.9
5	1.7		2.4	
Total	10.0	5.1	12.0	8.4
$\Sigma(Y_{ij})^2$	20.26	8.75	29.06	17.7

$G = Y_{..} = 35.5$

Step 2

Calculate the Correction Factor (CF) $= G^2/N = (Y_{..})^2/\Sigma r_i = 35.5^2/17 = 74.132$

Step 3

Calculate the Total SS

$$SS_{Total} = \Sigma(Y_{ij})^2 - CF = (2.0^2 + 2.2^2 + 1.8^2 + \dots + 1.9^2) - CF = 75.77 - 74.132$$

$$= 1.638$$

Step 4

Calculate $SS_{Treatment} = \Sigma\,(Y_{i.})^2/r_i - CF$

$$= \{(10)^2/5 + (5.1)^2/3 + (12)^2/5 + (8.4)^2/4\} - 74.132 = 0.978$$

Step 5

Calculate $SS_{Error} = SS_{Total} - SS_{Treatment} = 1.638 - 0.978 = 0.66$

Step 6

ANOVA Table for CRD

Sources of Variation	Sum of Squares (SS)	Degree of Freedom (d.f.)	Mean Sum of Square (MSS)	Variation Ratio (F_{cal})
Between Treatments	0.978	t - 1 = 4 - 1 = 3	$MS_{Treatment} = 0.978/3 = 0.326$	$MS_T/MS_E = 6.42$
Error	0.660	n - t = 17 - 4 = 13	$MS_{Error} = 0.660/13 = 0.05076$	
Total	1.638	n - 1 = 17 - 1 = 16		

Step 7

We find the following $F_{critical}$ values from F-table

$F_{0.05,3,13} = 3.41$ and $F_{0.01,3,13} = 5.74$

Step 8

Compare F_{cal} with $F_{critical}$ to draw inference –

(i) Since F_{cal} (6.42) > $F_{citical}$ (3.41) at 95% level of confidence we reject the null hypothesis.

(ii) Since F_{cal} (6.42) > $F_{citical}$ (5.74) at 99% level of confidence we reject the null hypothesis.

Randomized Block Design (RBD)

Suppose we want to compare the effects of t treatments, each treatment being replicated an equal number of times, say r times. Then we need n = rt experimental units, and these units are not perhaps homogeneous. The RBD consists of two steps. The first step is to divide the units into r more or less homogeneous groups. In each group or block we take as many units as there are treatments. Thus the number of blocks are the same as the common replication number (r). The same technique should be applied to the units of the block. Variation in technique, if any, should be made between the blocks. In agricultural field experiments sometimes a fertility

The second step is to assign the treatments at random to the units of a block. This is the difference of RBD from CBD. In an RBD randomization is restricted within a homogeneous block.

With this design each treatment will have the same number of

Randomized Block Design (RBD)

replication. If we want additional replications for some treatments, each of these may be applied to more than one unit in a block.

Let us consider five treatments – A, B, C, D and E replicated 4 times. We divide the whole experimental area into four relatively homogeneous strata or blocks and each block into 5 units or plots, treatments are then allocated at random to the plots of a block, fresh randomization done for each block. A particular layout may be as follows -

Block I	A	E	B	D	C
Block II	E	D	C	B	A
Block III	C	B	A	D	E
Block IV	A	D	E	C	B

Advantages of RBD **:** Chief advantages of RBD are as follows –

(i) Accuracy : This design has been shown to be more efficient or accurate than CRD for most types of experimental work. The elimination of between sum of squares from residual sum of squares usually results in a decrease of error mean sum of squares.
(ii) Flexibility : In RBD no restrictions are placed on the number of treatments or the number of replicates.
(iii) Ease of analysis : Statistical analysis is simple, rapid and straight forward.

Disadvantages of RBD **:** (i) RBD may give misleading results if blocks are not homogeneous.
(ii) RBD is not suitable for large number of treatments because in that case the block size will increase and it may not be possible to keep large blocks homogeneous.
(iii) If the data on more than two plots is missing the statistical analysis becomes tedious and complicated.

The analysis of RBD is exactly same as that of ANOVA-Two Way

If in an RBD, a single observation is made on each of the experimental units, then its analysis is same as ANOVA for fixed effect model for a two-way classified data with one observation per cell.

Null Hypothesis

H_{01} : Treatments are homogeneous ie. $\tau_1 = \tau_2 = \tau_3 = \dots = \tau_t$

H_{02} : the blocks are homogeneous ie. $b_1 = b_2 = b_3 = \dots = b_r$

Alternative Hypothesis

H_{11} : At least two τ_i's are different

H_{12} : At least two b_j's are different

The linear model is given by

$y_{ij} = \mu + \tau_i + b_j + e_{ij}$; where, $i = 1, 2, \dots, t$ and $j = 1, 2, \dots, r$)

Partitioning of the sum of squares

$(S_T)^2 = (S_t)^2 + (S_b)^2 + (S_e)^2$

$= \Sigma\Sigma(y_{ij} - \bar{y}_{..})^2$ is the total sum of squares $(S_T)^2$

$(S_t)^2 = \Sigma(\bar{y}_{i.} - \bar{y}_{..})^2$ is sum of squares due to treatments $(S_t)^2$

$(S_b)^2 = \Sigma(\bar{y}_{i.} - \bar{y}_{..})^2$ is sum of squares due to blocks $(S_b)^2$

$(S_e)^2 = \Sigma\Sigma(y_{ij} - \bar{y}_{i.} - \bar{y}_{.j} + \bar{y}_{..})^2$ is the sum of squares due to error $(S_e)^2$

$RSS = \Sigma Y^2$, $CF = G^2/n$

$TSS = RSS - CF$

$SST = \Sigma(T_i)^2/r - CF$

$SSRe(bl) = \Sigma(R_i)^2/t - CF$

$SSE = TSS - SST - SSR$

Partitioning of the sum of squares

$(S_T)^2 = (S_t)^2 + (S_b)^2 + (S_e)^2$

$= \Sigma\Sigma(y_{ij} - \overline{y}_{..})^2$ is the total sum of squares $(S_T)^2$

$(S_t)^2 = \Sigma(\overline{y}_{i.} - \overline{y}_{..})^2$ is sum of squares due to treatments $(S_t)^2$

$(S_b)^2 = \Sigma(\overline{y}_{i.} - \overline{y}_{..})^2$ is sum of squares due to blocks $(S_b)^2$

$(S_e)^2 = \Sigma\Sigma(y_{ij} - \overline{y}_{i.} - \overline{y}_{.j} + \overline{y}_{..})^2$ is the sum of squares due to error $(S_e)^2$

$RSS = \Sigma Y^2$, $CF = G^2/n$

$TSS = RSS - CF$

$SST = \Sigma(T_i)^2/r - CF$

$SSRe(bl) = \Sigma(R_i)^2/t - CF$

$SSE = TSS - SST - SSR$

ANOVA Table for RBD

Sources of Variation	Sum of Squares	Degree of Freedom (d.f.)	Mean Sum of Square	Variation Ratio (F_{cal})
Between Treatments	$(S_t)^2$	$(t - 1)$	$MSt = (S_t)^2/(t - 1)$	$F_t = MS_t/MS_e$
Blocks or replicates	$(S_b)^2$	$(r - 1)$	$MS_b = (S_b)^2/(r - 1)$	$F_b = MS_b/MS_e$
Error or residual	$(S_e)^2$	$(t - 1)(r - 1)$	$MSe = (S_e)^2/\{(t - 1)(r - 1)\}$	

Ex 2 **:** A randomized block design experiment is run with three treatment. The three treatmeans are – 6, 7 and 11. Total sum of squares is 220. ANOVA table takes the following form.

Source of Variations	DF	SS	MSS	F-ratio
Treatments				
Blocks		132		
Error				
Total	11	220		

Find all the missing entries in the ANOVA table and draw inference at LOS %.

Soln : Since, it is given that RBD is run with 3 treatment, so DF for treatments = 3 – 1 = 2. As the total df is 11, total number of observation will be 11 + 1 = 12 ie. number of blocks = 12/3 = 4. Hence DF for blocks = 4 – 1 = 3 and DF for error = 11 – 2 – 3 = 6.

Source of Variations	DF	SS	MSS = SS/DF	Treatment Means	Treatment toal (Mean x No. of treatments)
Treatments	2	56	56/2 =28	6	6x4 = 24
Blocks	3	132	132/3 = 44	7	7x4 = 28
Error	6	220 – (132 + 56) = 32	5.333	11	11x4 = 44
Total	11	220			96

We know SSTreatment = $\Sigma(T_i)^2/n_i - CF = (24^2 + 28^2 + 44^2)/4 - 768$

$= 824 - 768 = 56$

CF = $G^2/N = 96^2/3x4 = 9216/12 = 768$

Source of Variations	DF	SS	MSS = SS/DF	F_{ratio} = MSS/MSE	$F_{critical}$ (From F-table)
Treatments	2	56	28	5.25	$F_{0.05, 2, 6} = 5.14$
Blocks	3	132	44	8.25	$F_{0.05, 3, 6} = 4.76$
Error	6	32	5.333		
Total	11	220			

Conclusion **:** Now since, F_{ratio} for Treatments and Blocks are > $F_{critical}$ we cannot accept the null hypothesis.

Ex 3 **:** Test whether all 4 replicates (blocks) are equally effective and all 6 seeding rates (treatments) are equally effective in giving the yield from the following data:

Grain yield of rice at six seeding rates (Mg/Ha)

Replicate	Seeding rate (kg/ha)					
	25	50	75	100	125	150
1	5.1	5.3	5.3	5.2	4.8	5.3
2	5.4	6.0	5.7	4.8	4.8	4.5
3	5.3	4.7	5.5	5.0	4.4	4.9
4	4.7	4.3	4.7	4.4	4.7	4.1

Note **:** There are 4 rows & 6 columns

Soln :

H_{0t} **:** All six seeding rates (treatments) are equally effective.

H_{0b} **:** All four replicates (blocks) are equally effective.

Replicate	Seeding rate (kg/ha)						$Y_{.j}$
	25	50	75	100	125	150	
1	5.1	5.3	5.3	5.2	4.8	5.3	31
2	5.4	6.0	5.7	4.8	4.8	4.5	31.2
3	5.3	4.7	5.5	5.0	4.4	4.9	29.8
4	4.7	4.3	4.7	4.4	4.7	4.1	26.9
$Y_{i.}$	20.5	20.3	21.2	19.4	18.7	18.8	118.9
$\Sigma(Y_{ij})^2$	105.35	104.67	112.92	94.44	87.53	89.16	594.07

$= G = Y_{..}$

Step 1 Calculate Correction Factor (CF) = $(Y_{..})^2/(t\text{x}r) = 118.9^2/(6\text{x}4) = 589.05$

Step 2 Calculate Total SS = $\Sigma(Y_{ij})^2 – \text{CF}$

$= (5.1^2 + 5.4^2 + 5.3^2 + \ldots + 4.1^2) – 589.05 = 5.02$

Step 3 Calculate the Replicate SS (Rep SS) = $\Sigma(Y_{.j})^2/t – \text{CF}$

$= (31.0^2 + 31.2^2 + 29.8^2 + 26.9^2)/6 - 589.05 = 591.015 – 589.05 = 1.965$

Step 4 Calculate the Treatment SS (Trt SS) = $\Sigma(Y_{.j})^2/r - CF$

$= (20.5^2 + 20.3^2 + 21.2^2 + 19.4^2 + 18.7^2 + 18.8^2)/4 - 589.05 = 1.2675$

Step 5 Calculate the error SS = Total SS – Rep SS – Trt SS

$= 5.02 - 1.965 - 1.2675 = 1.7875$

Step 6

ANOVA Table

Sources of Variation	Sum of Squares	Degree of Freedom (d.f.)	Mean Sum of Square	Variation Ratio (F_{cal})
Between Replicates	1.965	r - 1 = 4 - 1 = 3	MSr = 1.965/3 = 0.655	$F_r = MS_r/MSe$ = 5.495
Between Treatments	1.2675	t - 1 = 6 - 1 = 5	$MS_t = 1.2675/5$ = 0.2535	$F_t = MS_t/Mse$ = 2.127
Error or residual	1.7875	(r - 1)(t - 1) = 15	MSe = 1.7875/15 = 0.1192	
Total	5.02	tr - 1 = 23		

Step 7 Obtain the $F_{critical}$ values from F-table for LOS (α) 0.05 and 0.01

For Replicate		For Treatment	
$F_{0.05, 3, 15}$	3.29	$F_{0.05, 5, 15}$	2.90
$F_{0.01, 3, 15}$	5.42	$F_{0.01, 5, 15}$	4.56

Step 8 Draw the inference -

Replicates **:** Since F_{calc} (5.495) > F_{tab} (3.29 & 5.42) at 95% & 99% levels of confidence, we reject the null hypothesis (ie. all replicate means are equal).

Treatment **:** Since F_{calc} (2.127) < F_{tab} (2.90 & 4.56) at 95% & 99% levels of confidence, we fail to reject the null hypothesis (ie. all treatment means are equal). In other words, we accept the null hypothesis.

Conclusion --

Replicates : Since F_{calc} (5.495) > F_{tab} (3.29 & 5.42) at 95% & 99% levels of confidence, we reject the null hypothesis (ie. all replicate means are equal).

Treatment : Since F_{calc} (2.127) < F_{tab} (2.90 & 4.56) at 95% & 99% levels of confidence, we fail to reject the null hypothesis (ie. all treatment means are equal). In other words, we accept the null hypothesis.

Efficiency of RBD over CRD

If $r_i = r$ for all $i = 1, 2, \ldots, t$ treatments, then for such data, we can apply both CRD and RBD for testing the equality of effects of treatments. In such case, RBD is more efficient than CRD and the relative efficiency of RBD over CRD is calculated as –

$$E = [r(t - 1)MSEr + (r - 1)MSRe(Bl)]/[(rt - 1)MSEr]$$

For the previous example 2,

$$E = [4(6 - 1)0.1192 + (4 - 1)0.655]/[(4x6 - 1)0.1192]$$

$$= (2.384 + 1.965)/2.7416 = 4.349/2.7416 = 1.586 \approx 1.59$$

Therefore, RBD is approximately 59% more efficient than CRD.

Note **:** $E > 1$ means more efficient, $E < 1$ means less efficient and $E = 1$ means equally efficient.

RBD with one missing observation

In RBD, if any one observation is missed then it is estimated as
X = (rR’ + tT’ – G’)/[(r - 1)(t - 1)] and correction factor is calculated as
B (Bias) = [R’ - (t - 1)X]^2/[t(t - 1)], where, r and t are number of replicates and treatments respectively and R’, T’, G’ are the corresponding replicate, treatment and grand totals.

After inserting X in the original data, all sum of squares are calculated as usual similar to normal RBD without the calculation of error sum of squares.

Corrected total sum of squares, corrected treatment sum of squares and corrected error sum of squares are calculated as
CTSS = TSS - 2B, CSST = SST - B, CSSE = CTSS – CSST - SSB

In the ANOVA table, both SST, SSE and TSS are replaced by CSST, CSSE and CTSS respectively and also error df is taken as [(r - 1)(t - 1) - 1] and total df is taken as (rt - 2).

Remaining procedure is as usual similar to normal RBD.

Ex 4 : Suppose that the value for treatment 2 is missing in replication III. The data will then be as presented in the table below.

Treatment (t = 5)	Replication (r = 4)				Total Treatment
	I	II	III	IV	
1	22.9	25.9	39.1	33.9	121.8
2	29.5	30.4	X missing = 33.1	29.6	89.5 (T’)
3	28.8	24.4	32.1	28.6	113.9
4	47	40.9	42.8	32.1	162.8
5	28.9	20.4	21.1	31.8	102.2
Total	157.1	142	135.1 (R’)	156	590.2 (G’)

T' = 89.5, R' = 135.1, G' = 590.2

H_{0t} : All treatment mean effects are equal and

H_{1t} : The treatment mean effects are not equal.

H_{0R} : All replicate means are homogeneous and

H_{1R} : All replicate means effects are not homogeneous

$X = (rR' + tT' - G')/\{(r-1)(t-1)\} = (4x135.1 + 5x89.5 - 590.2)/(3x4)$

$= (540.4 + 447.5 - 590.2)/12 = 33.1$

The upward Bias, $B = \{R' - (t-1)(X)\}^2/\{t(t-1)\}$

$= \{135.1 - (4x33.1)\}^2/(5x4) = 7.29/20 = 0.3645$

After substituting the estimated missing value, we get –

Treatment 2 total $(T' + X) = 89.5 + 33.1 = 122.6$

Replication 3 total $(R' + X) = 135.1 + 33.1 = 168.2$ and

The Grand total $(G' + X) = 590.2 + 33.1 = 623.3$

$T' = 89.5,\ R' = 135.1,\ G' = 590.2$
$t = 5,\ r = 4,\ CF = G^2/rt = 623.3^2/4x5$

Treatment SS $= \Sigma(T_i)^2/r - CF$

$= \{121.8^2 + 122.6^2 + 113.9^2 + 162.8^2 + 102.2^2\}/4 - 623.3^2/20$

$= 19946.9725 - 19425.1445 = 521.828$

Corrected Treatment SS (Treatment SS – B) $= 521.828 - 0.3645 = 521.4635$

TSS (including X)

$= \Sigma Y^2 - CF = (22.9^2 + 25.9^2 + \ldots + 33.1^2 + \ldots + 31.8^2) - 623.3^2/20$

$= 20364.47 - 19425.1445 = 939.3255$

Corrected TSS (TSS – 2B) = $939.3255 - 2\text{x}0.3645 = 938.5965$

SSB(Re) = $\Sigma(Ri)^2/t - CF = (157.1^2 + 142^2 + 168.2^2 + 156^2)/5 - 623.3^2/20$

$= 19494.33 - 19425.1445 = 69.1855$

Corrected SSE = Cor TSS – Cor SST – SSB (Re)

$= 938.5965 - 521.4635 - 69.1855 = 347.9475$

ANOVA Table (r = 4, t = 5)

Sources of Variation	Sum of Squares	Degree of Freedom (d.f.)	Mean Sum of Square	Variation Ratio (F_{cal})
Replication (Block)	69.1855	r - 1 = 3	MSr = 69.1855/3 = 23.0618	$F_r = MS_r/MSe$ = 23.0618/31.6316 = 0.729
Treatment (Corrected)	521.4635	t - 1 = 4	MS_t = 521.4635/4 = 130.3659	$F_t = MS_t/Mse$ =130.3659/31.6316 = 4.121
Error (Corrected)	347.9475	(r - 1)(t – 1) -1 = 11	MSe = 347.9475/11 = 31.6316	
Total (Corrected)	938.5965	tr – 2 = 18		

Conclusion : $F_{critical}$ at LOS α = 5% for DF 3,11 ie. $F_{0.05,3,11} = 3.59$ and $F_{cal} = 0.729$ for replication (block). Since $F_{cal} < F_{crit}$ H_0 for replicates is accepted ie. replicate effects are homogeneous.

$F_{critical}$ at LOS α = 5% for DF 4,11 ie. $F_{0.05,4,11} = 3.36$ and $F_{cal} = 4.121$ for Treatment. Since $F_{cal} > F_{crit}$ H_0 for treatment effect can not be accepted ie. treatment effects are not homogeneous.

Conclusion : $F_{critical}$ at LOS $\alpha = 5\%$ for DF 3,11 ie. $F_{0.05,3,11} = 3.59$ and $F_{cal} = 0.729$ for replication (block). Since $F_{cal} < F_{crit}$ H_0 for replicates is accepted ie. replicate effects are homogeneous.

$F_{critical}$ at LOS $\alpha = 5\%$ for DF 4,11 ie. $F_{0.05,4,11} = 3.36$ and $F_{cal} = 4.121$ for Treatment. Since $F_{cal} > F_{crit}$ H_0 for treatment effect can not be accepted ie. treatment effects are not homogeneous.

Practice Problems

Q1. A researcher reports the Relative Efficiency of a Randomized Block Design, relative to a Completely Randomized Design of 5. The Randomized Block Design had 5 treatments and 8 blocks. How many observations would be needed to have as precise of estimates of treatment means, if the experiment was conducted as to a Completely Randomized Design?

Q2. An experiment is conducted as a Randomized Block Design, with 10 blocks and 5 treatments, reports a relative efficiency (relative to a Completely Randomized Design) of RE = 5. How many TOTAL subjects would be needed if we ran a CRD, to obtain the same standard error of a treatment mean?

i) 50 ii) 10 iii) 25 iv) 250 v) 500

(i) Ascertain if it is One-way ANOVA or Two-way ANOVA?

(ii) Construct ANOVA Table.

Practice Problems

Q3. Given the following date, test the hypothesis.

H_0 : All the means are equal; H_1 : At least 2 means are different

$n_1 = 4, \bar{x}_1 = 27, s_1 = 4;\ n_2 = 7, \bar{x}_2 = 25, s_2 = 9;\ n_3 = 5, \bar{x}_3 = 28, s_3 = 5$

(i)Is it One-Way or Two-Way ANOVA?

(ii)Construct ANOVA Table.

BSDCHZC355
Statistical Inference & Applications

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

Shaibal Kr. Sen
Session 07

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

STATISTICAL INFERENCES

&

APPLICATIONS

Scheduled Topics to be covered

Design of Experiments **:** Complete Block Designs contd.

Latin Square Design (LSD)

Analysis of RBD and LSD with one missing observation

Comparison of efficiencies of CRD, RBD and LSD

Latin Square Design (LSD)

In LSD, blocking (local control) principle is applied in two perpendicular directions. The output times of a program for a common problem written by 3 different programmers - Krishna, Rani, Sathya (Rows) using 3 different software Python, R, SAS (Columns) executed in three different brands of laptops - Apple, Lenovo, Acer are given in the following design.

Time (in seconds)	Python	R	SAS
Krishna	Lenovo	Apple	Acer
	20	24	22
Rani	Apple	Acer	Lenovo
	23	26	28
Sathya	Acer	Lenovo	Apple
	20	27	29

Latin Square Design (LSD)

The interest is to test whether average output times due to three programs are equal, three software are equal and due to three different brands of Laptops are equal. Then LSD is the suitable design in this case.

We set up - Null Hypothesis -
H_{0R} : The average output times due to three programs are equal.
H_{0C} : The average output times due to three software are equal.
H_{0T} : The average output times due to three branded laptops are equal.

And Alternative Hypothesis -
H_{1R} : The average output times due to three programs are not equal.
H_{1C} : The average output times due to three software are not equal.
H_{1T} : The average output times due to three branded laptops are not equal.

LSD is defined for eliminating the variation of two factors called row and column in this design. The number of treatments is equal to number of replications. Thus in case of m treatments there have to be max$^{\underline{m}}$ m^2 experimental units. The whole of experimental area is divided into m^2 experimental units (plots) arranged in a square so that each row as well as each column contain m units.

The m treatments are allocated at random to these rows and columns in such a way that every treatment occurs only once in each row and in each column. Such a layout is LSD.

3 x 3 Layout		
A	B	C
B	C	A
C	A	B

Ex : An animal feeding experiment where the column groups may corresponding with initial weight and the row group with age.

Let us set up null hypothesis –

For row effects $H_{0r} : r_1 = r_2 = \dots = r_m = 0$

For column effects $H_{0c} : c_1 = c_2 = \dots = c_m = 0$

For treatment effects $H_{0t} : t_1 = t_2 = \dots = t_m = 0$

Alternative hypothesis –

For row effects H_{1r} : At least 2 r_i s are different

For column effects H_{1c} : At least 2 c_i s are different

For treatment effects H_{1t} : At least 2 t_i s are different

Let y_{ijk} (i, j, k = 1, 2, 3,…., m) denote the response from the unit in the $i^{\underline{th}}$ row, $j^{\underline{th}}$ column and receiving the $k^{\underline{th}}$ treatment.

The model is $y_{ijk} = \mu + r_i + c_j + t_k + e_{ijk}$; where i, j, k = 1, 2, 3,…., m

where μ is the constant mean effect; r_i, c_j and t_k are due to $i^{\underline{th}}$ row, $j^{\underline{th}}$ column and $k^{\underline{th}}$ treatment respectively and e_{ijk} is the error effect due to random component assumed to be normally distributed with mean zero and variance σ_e^2 ie. $e_{ijk} \sim N(0, \sigma_e^2)$.

If we write,

G = Total of all the m^2 observations.

Note **:** Total of all observations, $N = m^2$

R_i = Total of the m observations in the $i^{\underline{th}}$ row.

C_j = Total of the m observations in the $j^{\underline{th}}$ column.

T_k = Total of the m observations from the $k^{\underline{th}}$ treatment.

ANOVA Table

Source of Variation	DF	Sum of Squares	Mean Sum of Sqs	Variance Ratio (Fcal)
Rows	$m - 1$	$SSR = (S_R)^2$	$MSR = (S_R)^2/(m - 1)$	$FR = MSR/MSE$
Columns	$m - 1$	$SSC = (S_C)^2$	$MSC = (S_C)^2/(m - 1)$	$FC = MSC/MSE$
Treatments	$m - 1$	$SST = (S_T)^2$	$MST = (S_T)^2/(m - 1)$	$FT = MST/MSE$
Error	$(m - 1)(m - 2)$	$SSE = (S_E)^2$	$MSE = (S_E)^2/(m - 1)(m - 2)$	
Total	$m^2 - 1$	TSS		

Total sum of square (TSS)= $\sum_{ijk}(y_{ijk})^2 – G^2/m^2$

Row sum of square (SSR) = $(S_R)^2 = \sum^{i}(R_i)^2/m – G^2/m^2$

Column sum of square (SSC) = $(S_C)^2 = \sum^{j}(C_j)^2/m - G^2/m^2$

Treatment sum of square (SST) = $(S_T)^2 = \sum^{k}(T_k)^2/ m - G^2/m^2$

SSE = TSS – SSR – SSC – SST

Obtain the F_α value for DF = {(m – 1), (m – 1)(m – 2)} at the level of significance α. If $F_R > F_\alpha$ reject H_{0R} and if $F_R < F_\alpha$ can not reject H_{0R}. Similarly, it can be tested for H_{0C} and H_{0T}.

Also, if $p_R < \alpha$, then reject H_0 and if $p_R > \alpha$ accept H_0. Similarly, it can inferred for p_C and p_T.

Advantages of LSD –

With two-way grouping LSD controls more of the variation than CRD or RBD.

The two-way elimination of variation as a result of cross grouping often results in small error mean sum of squares.

LSD is an incomplete 3-way layout. Its advantage over the complete 3-way layout is that instead of m^3 experimental units only m^2 units are needed. Thus a 4x4 LSD results in saving $m^3 = 4^3 – 4^2 = 48$ observations over a complete 3-way layout.

The statistical analysis is simple though slightly complicated than for RBD. Even 1 or 2 missing observations the analysis remains relatively simple.

More than one factor can be investigated simultaneously.

Disadvantages of LSD –

LSD is suitable for the number of treatments between 5 and 10 and for more than 10 to 12 treatments the design is seldom used. Since in that case, the square becomes too large and does not remain homogeneous.

In case of missing plots the statistical analysis becomes quite complex.

If one or two blocks in a field are affected by some disease or pest, we can't omit because the number of rows columns and treatments have to be equal.

Ex 1 : Perform ANOVA of the following LSD.

An experiment was carried out to determine the effect of claying the ground on the field of barley grains; amount of clay used were as follows –

A : No clay.

B : Clay at 100 per acre.

C : Clay at 200 per acre.

D : Clay at 300 per acre.

The yields were in plots of 8 meters by 8 meters and are given in the table.

D 29.1	B 18.9	C 29.4	A 5.7
C 16.4	A 10.2	D 21.2	B 19.1
A 5.4	D 38.8	B 24.0	C 37.0
B 24.9	C 41.7	A 9.5	D 28.9

Soln :

H_{0r} : Row means are all equal - vs- H_{1r} : Row means are not all equal.

H_{0c} : Column means are all equal - vs- H_{1c} : Column means are not all equal.

H_{0t} : Treatment means are all equal - vs- H_{1t} : Treatment means are not all equal.

	I	II	III	IV	Row Totals (Ri)
I	D 29.1	B 18.9	C 29.4	A 5.7	83.1
II	C 16.4	A 10.2	D 21.2	B 19.1	66.9
III	A 5.4	D 38.8	B 24	C 37	105.2
IV	B 24.9	C 41.7	A 9.5	D 28.9	105
Column Totals (Cj)	75.8	109.6	84.1	90.7	360.2 = G

The four treatment totals are –

A : 30.8	B : 86.9	C : 124.5	D : 118.0
(5.7 + 10.2 + 5.4 + 9.5 = 30.8)	(18.9 + 19.1 + 24 + 24.9 = 86.9)	(29.4 + 16.4 + 37 + 41.7 = 124.5)	(29.1 + 21.2 + 38.8 + 28.9 = 118)

Grand Total $G = 360.2$, $N = m^2 = 16$; $CF = G^2/N = 360.2^2/16 = 8109.0025$

Raw SS (RSS) $= \Sigma(y_{ijk})^2 = 29.1^2 + 18.9^2 + \ldots.. + 9.5^2 + 28.9^2 = 10{,}052.08$

Total SS (TSS) $= RSS - CF = 10{,}052.08 - 8109.0025 = 1943.0775$

$SSR = \Sigma(R_i)^2/m - CF = (83.1^2 + 66.9^2 + 105.2^2 + 105^2)/4 - 8109.0025 = 259.3125$

$SSC = \Sigma(C_j)^2/m - CF = (75.8^2 + 109.6^2 + 84.1^2 + 90.7^2)/4 - 8109.0025 = 155.2725$

$SST = \Sigma(T_k)^2/m - CF = (30.8^2 + 86.9^2 + 124.5^2 + 118^2)/4 - 8109.0025 = 1372.1225$

Error SS $= TSS - SSR - SSC - SST$

$= 1943.0775 - 259.3125 - 155.2725 - 1372.1225 = 156.37$

ANOVA Table for LSD

Source of Variation	DF	SS	MSS = SS/DF	F_{cal} = MSS/MSE
Rows	m - 1 = 3	259.3125	86.4375	FR = 3.316
Columns	m - 1 = 3	155.2725	51.7575	FC = 1.986
Treatments	m - 1 = 3	1372.123	457.3742	FT = 17.549
Error	(m - 1)(m - 2) = 6	156.37	26.0616	
Total	m^2 - 1 = 15	1943.0775		

$F_{critical}$ value obtained from F-table for $\alpha = 0.05$ and df 3, 6 ie. $F_{0.05, 3, 6} = 4.76$. Now, comparing the F_{cal} with $F_{critical}$ we conclude that the null hypothesis for rows and columns can not be rejected whereas for treatments null hypothesis is rejected ie. different levels of clay have significant effect on the yield.

Rows	Columns	Treatments
F_{cal} (3.316) < $F_{critical}$ (4.76)	F_{cal} (1.986) < $F_{critical}$ (4.76)	F_{cal} (17.549) > $F_{critical}$ (4.76)
So, Null hypothesis can not be rejected.	So, Null hypothesis can not be rejected.	So, Null hypothesis can not be accepted.

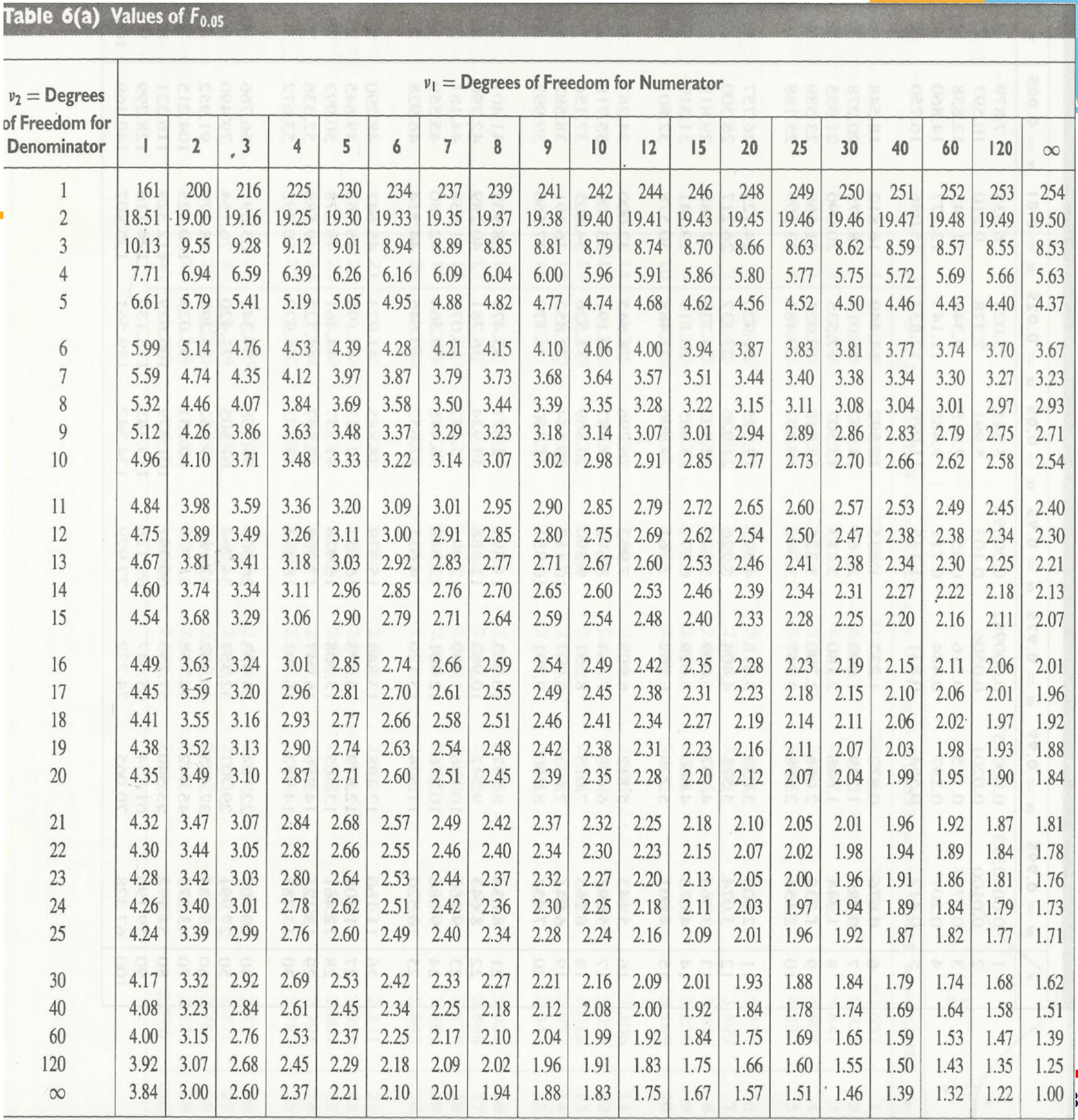


Table 6(a) Values of $F_{0.05}$

ν_2 = Degrees of Freedom for Denominator	ν_1 = Degrees of Freedom for Numerator																		
	1	2	3	4	5	6	7	8	9	10	12	15	20	25	30	40	60	120	∞
1	161	200	216	225	230	234	237	239	241	242	244	246	248	249	250	251	252	253	254
2	18.51	19.00	19.16	19.25	19.30	19.33	19.35	19.37	19.38	19.40	19.41	19.43	19.45	19.46	19.46	19.47	19.48	19.49	19.50
3	10.13	9.55	9.28	9.12	9.01	8.94	8.89	8.85	8.81	8.79	8.74	8.70	8.66	8.63	8.62	8.59	8.57	8.55	8.53
4	7.71	6.94	6.59	6.39	6.26	6.16	6.09	6.04	6.00	5.96	5.91	5.86	5.80	5.77	5.75	5.72	5.69	5.66	5.63
5	6.61	5.79	5.41	5.19	5.05	4.95	4.88	4.82	4.77	4.74	4.68	4.62	4.56	4.52	4.50	4.46	4.43	4.40	4.37
6	5.99	5.14	4.76	4.53	4.39	4.28	4.21	4.15	4.10	4.06	4.00	3.94	3.87	3.83	3.81	3.77	3.74	3.70	3.67
7	5.59	4.74	4.35	4.12	3.97	3.87	3.79	3.73	3.68	3.64	3.57	3.51	3.44	3.40	3.38	3.34	3.30	3.27	3.23
8	5.32	4.46	4.07	3.84	3.69	3.58	3.50	3.44	3.39	3.35	3.28	3.22	3.15	3.11	3.08	3.04	3.01	2.97	2.93
9	5.12	4.26	3.86	3.63	3.48	3.37	3.29	3.23	3.18	3.14	3.07	3.01	2.94	2.89	2.86	2.83	2.79	2.75	2.71
10	4.96	4.10	3.71	3.48	3.33	3.22	3.14	3.07	3.02	2.98	2.91	2.85	2.77	2.73	2.70	2.66	2.62	2.58	2.54
11	4.84	3.98	3.59	3.36	3.20	3.09	3.01	2.95	2.90	2.85	2.79	2.72	2.65	2.60	2.57	2.53	2.49	2.45	2.40
12	4.75	3.89	3.49	3.26	3.11	3.00	2.91	2.85	2.80	2.75	2.69	2.62	2.54	2.50	2.47	2.38	2.38	2.34	2.30
13	4.67	3.81	3.41	3.18	3.03	2.92	2.83	2.77	2.71	2.67	2.60	2.53	2.46	2.41	2.38	2.34	2.30	2.25	2.21
14	4.60	3.74	3.34	3.11	2.96	2.85	2.76	2.70	2.65	2.60	2.53	2.46	2.39	2.34	2.31	2.27	2.22	2.18	2.13
15	4.54	3.68	3.29	3.06	2.90	2.79	2.71	2.64	2.59	2.54	2.48	2.40	2.33	2.28	2.25	2.20	2.16	2.11	2.07
16	4.49	3.63	3.24	3.01	2.85	2.74	2.66	2.59	2.54	2.49	2.42	2.35	2.28	2.23	2.19	2.15	2.11	2.06	2.01
17	4.45	3.59	3.20	2.96	2.81	2.70	2.61	2.55	2.49	2.45	2.38	2.31	2.23	2.18	2.15	2.10	2.06	2.01	1.96
18	4.41	3.55	3.16	2.93	2.77	2.66	2.58	2.51	2.46	2.41	2.34	2.27	2.19	2.14	2.11	2.06	2.02	1.97	1.92
19	4.38	3.52	3.13	2.90	2.74	2.63	2.54	2.48	2.42	2.38	2.31	2.23	2.16	2.11	2.07	2.03	1.98	1.93	1.88
20	4.35	3.49	3.10	2.87	2.71	2.60	2.51	2.45	2.39	2.35	2.28	2.20	2.12	2.07	2.04	1.99	1.95	1.90	1.84
21	4.32	3.47	3.07	2.84	2.68	2.57	2.49	2.42	2.37	2.32	2.25	2.18	2.10	2.05	2.01	1.96	1.92	1.87	1.81
22	4.30	3.44	3.05	2.82	2.66	2.55	2.46	2.40	2.34	2.30	2.23	2.15	2.07	2.02	1.98	1.94	1.89	1.84	1.78
23	4.28	3.42	3.03	2.80	2.64	2.53	2.44	2.37	2.32	2.27	2.20	2.13	2.05	2.00	1.96	1.91	1.86	1.81	1.76
24	4.26	3.40	3.01	2.78	2.62	2.51	2.42	2.36	2.30	2.25	2.18	2.11	2.03	1.97	1.94	1.89	1.84	1.79	1.73
25	4.24	3.39	2.99	2.76	2.60	2.49	2.40	2.34	2.28	2.24	2.16	2.09	2.01	1.96	1.92	1.87	1.82	1.77	1.71
30	4.17	3.32	2.92	2.69	2.53	2.42	2.33	2.27	2.21	2.16	2.09	2.01	1.93	1.88	1.84	1.79	1.74	1.68	1.62
40	4.08	3.23	2.84	2.61	2.45	2.34	2.25	2.18	2.12	2.08	2.00	1.92	1.84	1.78	1.74	1.69	1.64	1.58	1.51
60	4.00	3.15	2.76	2.53	2.37	2.25	2.17	2.10	2.04	1.99	1.92	1.84	1.75	1.69	1.65	1.59	1.53	1.47	1.39
120	3.92	3.07	2.68	2.45	2.29	2.18	2.09	2.02	1.96	1.91	1.83	1.75	1.66	1.60	1.55	1.50	1.43	1.35	1.25
∞	3.84	3.00	2.60	2.37	2.21	2.10	2.01	1.94	1.88	1.83	1.75	1.67	1.57	1.51	1.46	1.39	1.32	1.22	1.00

One missing observation in LSD

Let us suppose that in [mxm] LSD, the observation occurring in the ith row, jth column and receiving the kth treatment is missing. Let's assume that its value is x, ie. $y_{ijk} = x$. Now let -

R_i' = Total of the known observations in the ith row.

C_j' = Total of the known observations in the jth column.

T_k' = Total of the known observations receiving kth treatment.

G' = Grand Total

$$X = \{m(R_i' + C_j' + T_k') - 2G'\}/\{(m-1)(m-2)\}$$

Ex 2 **:** In the following data, one value is missing. Estimate this value and analyse the given data.

A 12	C 19	B 10	D 8
C 18	B 12	D 6	A 7
B 22	D X	A 5	C 21
D 12	A 7	C 27	B 17

Soln :

Col Row	I	II	III	IV	Row Totals (Ri)	
I	A 12	C 19	B 10	D 8	49	
II	C 18	B 12	D 6	A 7	43	
III	B 22	D X	A 5	C 21	48 + X	R_3' = 48
IV	D 12	A 7	C 27	B 17	63	
Column Totals (C_j)	64	38 + X	48	53	203 + X	G' = 203
		C_2' = 38				

D = 8 + 6 + X + 12 = 26 + X $\quad$ T_4' = 26

Here, m = 4, R_3' = 48, C_2' = 38, T_4' = 26, G' = 203

Applying the missing estimation formula :

$$X = \{m(R_3' + C_2' + T_4') - 2G'\}/\{(m-1)(m-2)\} = \{4(48 + 38 + 26) - 2\text{x}203\}/3\text{x}2 = 7$$

Inserting the estimated value of X, we get the following observations -

Col / Row	I	II	III	IV	Row Totals (Ri)
I	A 12	C 19	B 10	D 8	49
II	C 18	B 12	D 6	A 7	43
III	B 22	D 7	A 5	C 21	55
IV	D 12	A 7	C 27	B 17	63
Column Totals (C_i)	64	45	48	53	G = 210

The four treatment totals are –

A (T_1) : 31

12 + 7 + 5 + 7 = 31

B (T_2) : 61

10 + 12 + 22 + 17 = 61

C (T_3) : 85

19 + 18 + 21 + 27 = 85

D (T_4) : 33

8 + 6 + 7 + 12 = 33

$CF = G^2/m^2 = 210^2/4^2 = 44100/16 = 2756.25$

Raw SS(RSS) $= \Sigma(y_{ijk})^2 = 12^2 + 18^2 + \ldots.. + 21^2 + 17^2 = 3432$

Total SS (TSS) = RSS – CF $= 3432 - 2756.25 = 675.75$

Row Sum of Squares (SSR) $= \Sigma(R_i)^2/m - CF$

$$= (49^2 + 43^2 + 55^2 + 63^2)/4 - 2756.25 = 54.75$$

Column Sum of Squares (SSC) $= \Sigma(C_j)^2/m - CF$

$$= (64^2 + 45^2 + 48^2 + 53^2)/4 - 2756.25 = 52.25$$

Sum of Squares of Treatment (SST) $= \Sigma(T_k)^2/m - CF$

$$= (\underset{(A)}{31^2} + \underset{(B)}{61^2} + \underset{(C)}{85^2} + \underset{(D)}{33^2})/4 - 2756.25 = 417.75$$

Error SS (SSE) = TSS – SSR – SSC – SST

$$= 675.75 - 54.75 - 52.25 - 417.75 = 151$$

ANOVA Table

Source of Variation	DF	SS	MSS = SS/DF	Variance Ratio		Conclusion
				Calculated	Tabulated $F_{0.05,3,5}$	
Rows	4 - 1 = 3	54.75	18.25	0.60	5.41	Insignificant
Columns	4 - 1 = 3	52.25	17.42	0.58	5.41	Insignificant
Treatments	4 - 1 = 3	417.75	139.25	4.61	5.41	Insignificant
Error	(m - 1)(m - 2) - 1 = 5	151	30.25			
Total	14	675.75				

Remark 1 **:** Efficiency of LSD over RBD

There may be two cases to judge the relative efficiency of LSD over RBD –

1. Relative efficiency of LSD over RBD, when rows are taken as blocks is

$$= \{MSSC + (m - 1)MSSE\}/(m \times MSSE)$$

where, MSSC = Mean Sum of Squares due to columns &
MSSE = Mean Sum of Squares due to error

2. Relative efficiency of LSD over RBD, when columns are taken as blocks is

$$= \{MSSR + (m - 1)MSSE\}/(m \times MSSE)$$

where, MSSR = Mean Sum of Squares due to rows, m = no. of rows or no. of treatments.

Remark 2 **:** Efficiency of LSD over CRD

Relative efficiency of LSD over CRD is given by

$$= \{MSSR + MSSC + (m - 1)MSSE\}/\{(m + 1)MSSE\}$$

To calculate efficiency of LSD over RBD and CRD with respect to

In example 1, m = 4, MSSR = 86.44, MSSC = 51.76, MSSE = 26.06
Relative efficiency of LSD over RBD

If rows are assumed as blocks then e = {MSSC + (m – 1)MSSE}/(mxMSSE)
= (51.76 + (4 - 1)x26.06)/(4*26.06) = 1.25 then 25% more efficient than RBD.

If columns are assumed as blocks then
e = {MSSR + (m - 1)MSSE}/(mxMSSE)
= (86.44 + (4 - 1)x26.06)/(4x26.06) = 1.58 then 58% more efficient than RBD.

Relative efficiency of LSD over CRD
e = {MSSR + MSSC + (m – 1)MSSE}/{(m + 1)MSSE}
= {51.76 + 86.44 + (4 - 1)x26.06}/{(4 + 1)x26.06} = 1.66 then 66% more efficient than CRD.

Ex : Perform ANOVA for the following LSD and obtain the relative efficiencies of the design over RBD and CRD. Use 0.01 level of significance.

Time(in seconds)	Python	R	SAS
Krishna	Lenovo 20	Apple 24	Acer 22
Rani	Apple 23	Acer 26	Lenovo 28
Sathya	Acer 20	Lenovo 27	Apple 29

H_{0R} : The three programme instructions are equally effective.
H_{1R} : The three programme instructions are not equally effective.

H_{0C} : The three softwares are equally effective.
H_{1C} : The three softwares are not equally effective.

H_{0T} : The three brands of computers are equally effective.
H_{1T} : The three brands of computers are not equally effective.

				Total (T_k)
T_1	0	8	7	15
T_2	4	3	9	16
T_3	2	6	0	8

	C_1	C_2	C_3
R_1	20	24	22
R_2	23	26	28
R_3	20	27	29

$m = 3,\ N = m^2 = 3^2 = 9$

$Y = 20$

	C_1	C_2	C_3	Total (R_i)
R_1	0 T_1	4 T_2	2 T_3	6
R_2	3 T_2	6 T_3	8 T_1	17
R_3	0 T_3	7 T_1	9 T_2	16
Total (C_j)	3	17	19	39 (G)

$$RSS = \Sigma Y^2 = 0^2 + 4^2 + 2^2 + 3^2 + 6^2 + 8^2 + 0^2 + 7^2 + 9^2 = 259$$

$$CF = G^2/N = 39^2/3^2 = (39/3)^2 = 13^2 = 169$$

$$TSS = RSS – CF = 259 – 169 = 90$$

$$SSR = \Sigma(R_i)^2/m – CF = (6^2 + 17^2 + 16^2)/3 – 169 = 193.66 – 169 = 24.66$$

$$SSC = \Sigma(C_j)^2/m – CF = (3^2 + 17^2 + 19^2)/3 – 169 = 184.66 – 169 = 15.66$$

$$SST = \Sigma(T_k)^2/m – CF = (15^2 + 16^2 + 8^2)/3 – 169 = 181.66 – 169 = 12.66$$

$$SSE = TSS – SSR – SSC – SST = 90 – 24.66 – 15.66 – 12.66 = 37.02$$

ANOVA Table for LSD

SV	SS	DF	MS = SS/DF	Variance Ratio (Fcal)	$F_{critical}$ $(F_{0.01, 2, 2})$
Rows	24.66	m - 1 = 2	24.66/2 = 12.33	MSR/MSE = 12.33/18.51 = 0.666	99
Columns	15.66	m - 1 = 2	15.66/2 = 7.83	MSC/MSE = 7.83/18.51 = 0.423	99
Treatments	12.66	m - 1 = 2	12.66/2 = 6.33	MST/MSE = 6.33/18.51 = 0.342	99
Error	37.02	(m - 1)(m - 2) = 2	37.02/2 = 18.51		
Total	90	m^2 - 1 = 8			

Since the calculated values of F for Rows, Columns and Treatments are < $F_{critical}$ values as depicted in the above ANOVA table, all the null hypothesis are accepted.

Efficiency of LSD

(i) Efficiency of LSD over RBD if rows are blocks –

e = {MSC + (m – 1)MSE}/mxMSE = (7.83 + 2x18.51)/3x18.51 = 0.8 < 1

ie. LSD is less efficient than RBD by 20%.

(ii) Efficiency of LSD over RBD if columns are blocks –

e = {MSR + (m – 1)MSE}/mxMSE = (12.33 + 2x18.51)/3x18.51 = 0.88 < 1

ie. LSD is less efficient than RBD by 12%

(iii) Efficiency of LSD over CRD if columns are blocks –

e = {MSSR + MSSC + (m – 1)MSSE}/{(m + 1)MSSE} = (12.33 + 7.83 2x18.51)/4x18.51 = 0.77 < 1

ie. LSD is less efficient than CRD by 23%

Miscellaneous

1. An experimenter randomly allocated 125 male turkeys to five treatment groups: control and treatments A, B, C, and D. There were 25 birds in each group, and the mean results were 2.16, 2.45, 2.91, 3.00, and 2.71, respectively. The sum of squares for experimental error was 153.4. Test the null hypothesis that the five group means are the same against the alternative that one or more of the treatments differs from the control.

Soln : $H_0 : \mu_A = \mu_B = \mu_C = \mu_D = \mu_{\text{control mean}}$

H_1 : At least one or more treatment means are not equal to control mean.

Given $n_1 = n_2 = n_3 = n_4 = n_5 = 25; N = 5x25 = 125, k = 5$

$\overline{x}_1 = 2.16, \overline{x}_2 = 2.45, \overline{x}_3 = 2.91, \overline{x}_4 = 3, \overline{x}_5 = 2.71$

$T_1 = n_1(\overline{x}_1) = 54, T_2 = n_2(\overline{x}_2) = 61.25, T_3 = n_3(\overline{x}_3) = 72.75,$

$T_4 = n_4(\overline{x}_4) = 75, T_5 = n_5(\overline{x}_5) = 67.75$

Given, SSE = 153.4. By CRD (ANOVA One-way),

$CF = G^2/N$, where $G = 25(2.16 + 2.45 + 2.91 + 3 + 2.71) = 330.75;$

and $N = 5x25 = 125$

$CF = 330.75^2/125 = 875.1645$

$$SST = \Sigma(T_i)^2/n_i - CF = \{54^2/25 + 61.25^2/25 + 72.75^2/25 + 75^2/25 + 67.75^2/25\}$$

$$-870.1645 = 887.0075 - 875.1645 = 11.843$$

Soln : $H_0 : \mu_A = \mu_B = \mu_C = \mu_D = \mu_{\text{control mean}}$

H_1 : At least one or more treatment means are not equal to control mean.

Given $n_1 = n_2 = n_3 = n_4 = n_5 = 25$; N = 5x25 = 125, k = 5

$\bar{x}_1 = 2.16$, $\bar{x}_2 = 2.45$, $\bar{x}_3 = 2.91$, $\bar{x}_4 = 3$, $\bar{x}_5 = 2.71$

$T_1 = n_1(\bar{x}_1) = 54$, $T_2 = n_2(\bar{x}_2) = 61.25$, $T_3 = n_3(\bar{x}_3) = 72.75$,

$T_4 = n_4(\bar{x}_4) = 75$, $T_5 = n_5(\bar{x}_5) = 67.75$

Given, SSE = 153.4. By CRD (ANOVA One-way),

$CF = G^2/N$, where $G = 25(2.16 + 2.45 + 2.91 + 3 + 2.71) = 330.75$;

and $N = 5x25 = 125$

$CF = 330.75^2/125 = 875.1645$

$SST = \Sigma(T_i)^2/n_i - CF = \{54^2/25 + 61.25^2/25 + 72.75^2/25 + 75^2/25 + 67.75^2/25\}$

$- 870.1645 = 887.0075 - 875.1645 = 11.843$

ANOVA One-Way Table for CRD

SV	SS	DF	MS = SS/DF	F_{cal}	$F_{critical}$ $(F_{0.02,4,120}$
Treatment	11.843	k - 1 =4	11.843/ 4 = 2.96	2.316	2.45
Error	153.4	n - k = 125 - 5 = 120	153.4/120 = 1.278		
Total	165.243	124			

Since $F_{cal} < F_{critical}$, we accept the null hypothesis.

2. The leaves of certain plants in the genus Albizzia will fold and unfold in various light conditions. We have taken fifteen different leaves and subjected them to red light for 3 minutes. The leaves were divided into three groups of five at random. The leaflet angles were then measured 30, 45, and 60 minutes after light exposure in the three groups. Data from W. Hughes. Analyze these data to test the null hypothesis that delay after exposure does not affect leaflet angle.

Delay (mins)	Angles (degrees)				
30	140	138	140	138	142
45	140	150	120	128	130
60	118	130	128	118	118

T_i	T_i^2	
T_1	T_1^2	$n_1 = 5$
T_2	T_2^2	$n_2 = 5$
T_3	T_3^2	$n_3 = 5$
G		N = 15

CRD ANOVA One-Way

H_0 : The three types delay does not effect the leaflet angles $\mu_{30} = \mu_{45} = \mu_{60}$

$TSS = RSS – CG = \Sigma Y^2 – G^2/N$

$SST = \Sigma(T_i)^2/n_i – CF = (T_1)^2/n_1 + (T_2)^2/n_2 + (T_3)^2/n_3 – G^2/N$

$SSE = TSS - SST$

SV	SS	DF	MS = SS/DF	F_{cal}	$F_{0.05,2,12}$
Treatments		k - 1 = 2	MSTr	MSTr/MSE	3.89
Error		n - k = 12	MSE		
Total		14			

3. Three different methods of analysis M1, M2, M3 are used to determine of a certain constituent in the sample. Each method is used by five analysis in the results, and the results are given in the results are given in table. Do these results indicate a significant variation either between the methods or between the analysts?

	Methods		
Analysts	M1	M2	M3
1	7.5	7.0	7.1
2	7.4	7.2	6.7
3	7.3	7.0	6.9
4	7.6	7.2	6.8
5	7.4	7.1	6.9

ANOVA-I way

Three different techniques namely – Medication, Exercises and Special Diet are randomly assigned to (individuals diagnosed with high blood pressure) to lower the blood pressure. After four weeks the reduction in each person is recorded. Test at 5% level, whether there is significant difference in mean reduction of blood pressure among three techniques.

Medication	10	12	9	15	13
Exercise	6	8	3	0	2
Diet	5	9	12	8	4

Soln :

Step 1 : Hypotheses

Null Hypothesis : $H_0 : \mu_1 = \mu_2 = \mu_3$

That is, there is no significant difference among the three groups on the average reduction in blood pressure.

Alternative hypothesis : $H_1 : \mu_1 \neq \mu_2$ for at least one pair (i, j); i, j = 1, 2, 3; i ≠ j

That is, there is significant difference in the average reduction in blood pressure in at least one pair of treatments.

Step 2 : Data

Medication	10	12	9	15	13
Exercise	6	8	3	0	2
Diet	5	9	12	8	4

Step 3 : Level of significance $\alpha = 0.05$

Step 4 : Test statistic, $F_0 = MST/MSE$

Step 5 : Calculation of Test Statistics

						Total	Square
Medication	10	12	9	15	13	59	3481
Exercise	6	8	3	0	2	19	361
Diet	5	9	12	8	4	38	1444
						G = 116	5286

Individual Squares

Medication	100	144	81	225	169
Exercise	36	64	9	0	4
Diet	25	81	144	64	16

$\Sigma\Sigma\,(x_{ij})^2 = 10^2 + 12^2 + 9^2 + 15^2 + 13^2 + 6^2 + 8^2 + 3^2 + 2^2 + 5^2 + 9^2 + 12^2 + 8^2 + 4^2 = 1162;$

1. Correction Factor (CF) = $G^2/N = 116^2/15 = = 13456/15 = 897.06$

2. Total Sum of Squares (TSS) **:** $\Sigma\Sigma(x_{ij})^2$ – CF = 1162 – 897.06 = 264.94

3. Sum of Squares between Treatments (SST) **:** $\Sigma\Sigma(x_i)^2/n_i$ – CF

= 5286/5 – 897.06 = 1057.2 – 897.06 = 160.14

4. Sum of Squares due to Error (SSE) **:** TSS – SST = 264.94 – 160.14 = 104.8

ANOVA Table (One-way)

SV	SS	DF	MS = SS/DF	F_{ratio}	$F_{crit} = F_{0.05,\ 2,12}$
Between Treatments	160.14	3 - 1 = 2	160.14/2 = 80.07	MSTr/MSE = 9.17	3.89
Error	104.8	12	104.8/12 = 8.73		
Total	264.94	n - 1 = 15 - 1 =14			

Step 6 **:** Critical value $F_{0.05,\ 2,12}$ = 3.89

Step 7 **:** Decision – As F_{ratio} (9.17) > $F_{critical}$ (3.89), the null hypothesis is rejected. Hence, we conclude that there exists significant difference in the reduction of the average blood pressure in at least one pair of techniques.

ANOVA II Way

A reputed marketing agency in India has three different training programmes for its salemen. The three programmes are Method – A, B, C. To assess the success of the programmes, 4 salesmen from each programmes were sent to the field. Their performance in terms of sales are given in the table below -

Salesmen	Methods		
	A	B	C
1	4	6	2
2	6	10	6
3	5	7	4
4	7	5	4

Test whether there is significant difference among methods and among salesmen.

Soln :

Step 1 : Hypotheses

Null Hypotheses : H_{01} : $\mu_{M1} = \mu_{M2} = \mu_{M3}$ (for methods / treatments)

That is, there is no significant difference among the three programmes in their mean sales.

H_{02} : $\mu_{s1} = \mu_{s2} = \mu_{s3} = \mu_{s4}$ (for blocks)

Alternative Hypotheses :

H_{11} : At least one average is different from the other, among the three programmes.

H_{12} : At least one average is different from the other, among the four salesmen.

Step 2 : Data

Salesmen	Methods		
	A	B	C
1	4	6	2
2	6	10	6
3	5	7	4
4	7	5	4

Step 3 : Level of significance $\alpha = 5\%$

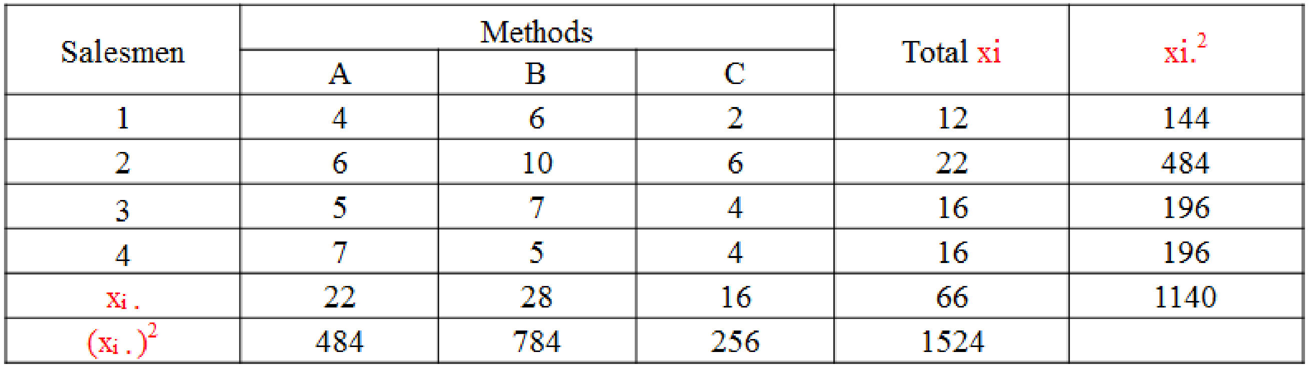
Step 4 : Test statistic : F_{ot}(treatment) = MST/MSE & F_{ob}(blocks) = MSB/MSE

Step 5 : Calculation of Test statistic

Salesmen	Methods			Total xi	xi.^{2}
	A	B	C		
1	4	6	2	12	144
2	6	10	6	22	484
3	5	7	4	16	196
4	7	5	4	16	196
$x_{i.}$	22	28	16	66	1140
$(x_{i.})^2$	484	784	256	1524	

Squares

16	36	4
36	100	36
25	49	16
49	25	16
		$\Sigma\Sigma(x_{ij})^2 = 408$



Salesmen	Methods			Total xi	xi.^{2}
	A	B	C		
1	4	6	2	12	144
2	6	10	6	22	484
3	5	7	4	16	196
4	7	5	4	16	196
$x_{i\,.}$	22	28	16	66	1140
$(x_{i\,.})^2$	484	784	256	1524	

Correction Factor (CF) = $G^2/N = 66^2/(3x4) = 4356/12 = 363$

Total Sum of Squares (TSS) = $\Sigma\Sigma\,(x_{ij})^2 - CF = 408 - 363 = 45$

Sum of squares due to treatment (SST) = $\sum_{i=1}^{k}(x_{\,.\,j})^2/k - CF = 1140/3 - 363 = 17$

Sum of Squares due to Blocks (SSB) = $\Sigma (x_{.j})^2/k$ – CF = 1524/4 – 363

= 381 – 363 = 18

Sum of Squares due to Error (SSE) = TSS – SST – SSB = 45 – 17 - 18 = 10

Mean Sum of Squares **:** MST = SST/(k – 1) = 17/3 = 5.67

MSB = SSB/(m – 1) = 18/2 = 9

MSE = SSE/(k – 1)(m – 1) = 10/6 = 1.67

ANOVA Table (Two-way)

SV	SS	DF	MS = SS/DF	Fratio	Fcritical
Between Treatments (Programmes)	17	4 - 1 = 3	17/3 = 5.67	MSTr/MSE = 3.4	$F_{0.05,3,6}$ = 4.76
Between Blocks (Salesmen)	18	3 - 1 = 2	18/2 = 9	MSB/MSE = 5.39	$F_{0.05,2,6}$ = 5.14
Error	10	6	10/6 = 1.67		
Total	45	11			

Step 6 : Critical values

$F_{0.05,3,6}$ = 4.76 (for treatments)

$F_{0.05,2,6}$ = 5.14 (for blocks)

Step 7 : Decision

(i) Calculated $F_{0t}(3.4) < F_{0.05,3,6}$ (4.76), the null hypothesis is not rejected and we conclude that there is significant difference in the mean sales among the three programmes.

(ii) Calculated $F_{0b}(5.39) > F_{0.05,2,6}$ (5.14), the null hypothesis is rejected and we conclude that there does not exist significant difference in the mean sales among the four salesmen.

Conclusion --

For Treatments : Since $F_{0t}(3.4) < F_{0.05,3,6}$ (4.76), the null hypothesis is not rejected and we conclude that there is significant difference in the mean sales among the three programmes.

For Blocks : Since Calculated $F_{0b}(5.39) > F_{0.05,2,6}$ (5.14), the null hypothesis is rejected and we conclude that there does not exist significant difference in the mean sales among the four salesmen.

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

BSDCHZC355
Statistical Inference & Applications

Shaibal Kr. Sen
Session 09

innovate achieve lead

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

STATISTICAL INFERENCES

&

APPLICATIONS

Scheduled Topics to be covered

Design of Experiments : Incomplete Block Designs

Balanced Incomplete Block Design (BIBD)

Partially Balanced Incomplete Block Design (PBIBD)

Incomplete Block Designs

Randomised Block Design (RBD) is a complete block design in which each block size (number of experimental units or plots in a block) is same as the number of treatments and the number of blocks or replicates is less than or more than or equal to number of treatments.

On the other hand an Incomplete Block Design is such that each block size (number of experimental units or plots) is less than the number of treatments and the number of blocks is more than or equal to number of treatments.

These designs were introduced by Yates in order to eliminate heterogeneity to a greater extent than is possible with Randomized Blocks and Latin Squares when the number of treatments is large. The precision of the estimate of a treatment effect depends on the number of replications of the treatment – if larger is the number of replications, the more is the precision. Similar is the case for the precision of estimate of the difference between two treatment effects. If a pair of treatment occurs together more number of times in the design, the difference between these two treatment effects can be estimated with more precision. To ensure equal or nearly equal precision of comparisons of different pairs of treatment effects, the treatments are so allocated to the experimental units in different blocks of equal sizes such that each treatment occurs at most once in a block and it has an equal number of replications and each pair of treatments has the same or nearly the same number of replications. When the number of replications of all pairs of treatments in a design is the same, then we have an important class of designs called Balanced Incomplete Block Designs (BIBD) and when there are unequal number of replications for different pairs of treatments, then the designs are called as Partially Balanced Incomplete Block Design (PBIBD).

Balanced Incomplete Block Designs (BIBDs) are formed mainly when number of plots in each block is less than the number of treatments.

A BIBD is an arrangement of v treatments in b blocks each of size k (< v) such that –

(i)Each treatment occurs at most once in a block.

(ii) Each treatment occurs in exactly r blocks.

(iii) Each pair of treatments occurs together in exactly λ block.

Hence, the parameters of the design are v = treatments, b = blocks, block size = k < v, Each treatment occurs in exactly r blocks, each pair of treatments occur in exactly λ block.

Ex : A BIBD for $v = b = 5$, $r = k = 4$ and $\lambda = 3$ in the following -

Blocks	(I)	1	2	3	4
	(II)	1	2	3	5
	(III)	1	2	4	5
	(IV)	1	3	4	5
	(V)	2	3	4	5

The symbols v, b, r, k, λ are called the parameters of the design. These parameters satisfy the relations : $vr = bk$…..(i) and $\lambda(v - 1) = r(k - 1)$…..(ii) where, $b \geq v$.

Each side of equation (i) represents the total number of experimental units or plots in the design. Equation (ii) can be established by noting that a given treatment occurs with $(k - 1)$ other treatments in each of r blocks and also occurs with each of the other $(v - 1)$ treatments in λ blocks.

Statistical Analysis

Consider the model : $y_{ij} = \mu + \tau_i + \beta_j + e_{ij}$

Observation = General Mean + Treatment Effect + Block Effect + Random Error

Random errors are assumed to be independently and identically distributed normally with mean zero and constant variance σ^2. On minimizing the error sum of squares with respect to the parameters, we get a set of normal equations which can be solved to get the estimates of different contrasts of various treatment & block effects.

Now we compute -

G = Grand Total of observations.

$\bar{y}$ = Grand Mean = G/n, where $n = vr = bk$ = Total No. of observations.

T_i = Sum of observations for treatment i, $(i = 1, 2, \ldots, v)$

B_j = Sum of observations in block j, $(j = 1, 2, \ldots, b)$

$CF = G^2/n$

Q_i = Adjusted ith treatment total

$= T_i$ – (Sum of block totals in which treatment i occurs)/Block size (k)

A solution for the ith treatment effect is, $\hat{\tau}_i = (kQ_i)/(\lambda v)$ $\quad (i = 1, 2, \ldots, v)$

Adjusted treatment mean for treatment i = ith treatment effects ($\hat{\tau}_i$) + grand mean ($\bar{y}$)

Various sums of squares can be obtained as follows –

(i) Total Sum of Squares (TSS) = $\Sigma(\text{observation})^2$ - CF

(ii) Treatment Sum of Squares unadjusted (SST_u) = $[\Sigma(T_i)^2]/r$ - CF

(iii) Block Sum of Squares unadjusted (SSBu) = $[\Sigma(B_j)^2]/k$ – CF

(iv) Treatment Sum of Squares adjusted (SST_A) = $\Sigma\, \hat{\tau}_i\, Q_i$

(v) Error Sum of Squares (SSE) = TSS – SSB_U - SST_A

(vi) Blocks Sum of Squares adjusted (SSB_A) = $SST_A + SSB_U - SST_U$

Analysis of Variance for a BIB is given in next slide -

ANOVA for BIB (v, b, r, k, λ) Design

Source of Vaiation	DF	SS	MS	$F_{\text{calculated}}$	F_{critical}
SSTreatment (unadjusted)	v - 1	SST_U			
SSBlocks (unadjusted)	b - 1	SSB_U			
SSTreatment (adjusted)	v - 1	SST_A	MST	MST/MSE	$F_{(v-1),(n-v-b+1)}$
SSBlocks (adjusted)	b - 1	SSB_A	MSB	MSB/MSE	$F_{(b-1),(n-v-b+1)}$
SSError	n - v - b + 1	SSE	MSE		
SSTotal	n - 1	TSS			

Ex : The following data relate to an experiment conducted using a BIB design with parameters v = 4, b = 4, r = 3, k = 3, λ = 2. The layout plan and yield figures (in coded units) are tabulated below -

Block Number	Treatments & Yield figures		
1	(1) 77	(2) 85	(3) 60
2	(1) 70	(2) 67	(4) 54
3	(1) 69	(3) 62	(4) 40
4	(2) 72	(3) 63	(4) 55

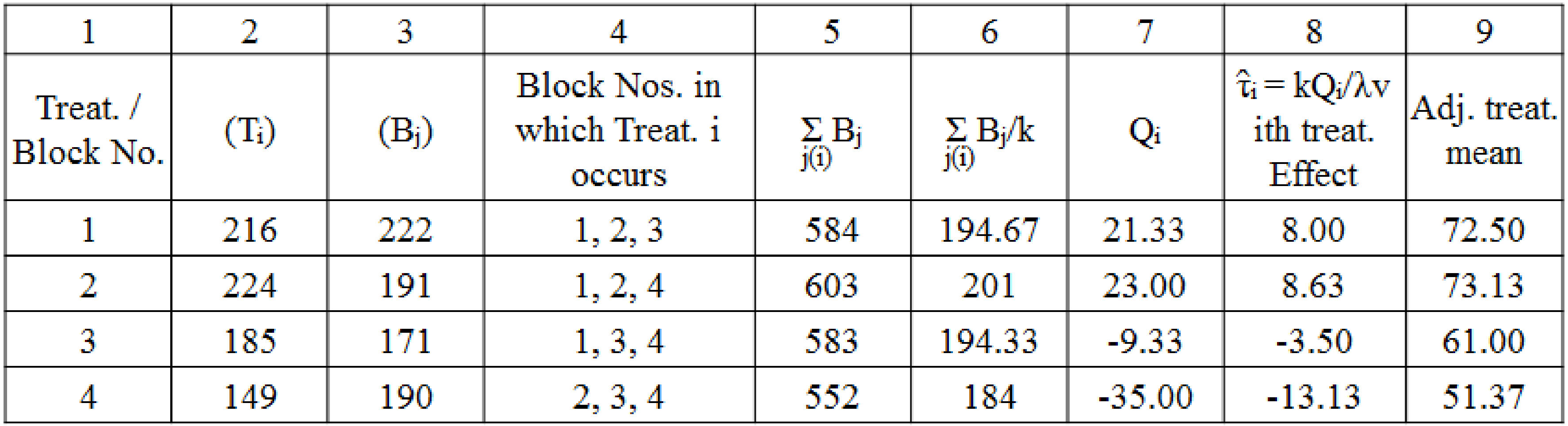
Carry out the analysis.

Soln : Grand Total, G = 77 + 85 + 60 + ….. + 55 = 774

No. of observations, n = 12, Grand mean, $\overline{y} = G/n = 774/12 = 64.5$

No. of replication, r = 3, Block size, k = 3, CF = G^2/n = 599076/12

= 49923



1	2	3	4	5	6	7	8	9
Treat. / Block No.	(T_i)	(B_j)	Block Nos. in which Treat. i occurs	$\sum_{j(i)} B_j$	$\sum_{j(i)} B_j/k$	Q_i	$\hat{\tau}_i = kQ_i/\lambda v$ ith treat. Effect	Adj. treat. mean
1	216	222	1, 2, 3	584	194.67	21.33	8.00	72.50
2	224	191	1, 2, 4	603	201	23.00	8.63	73.13
3	185	171	1, 3, 4	583	194.33	-9.33	-3.50	61.00
4	149	190	2, 3, 4	552	184	-35.00	-13.13	51.37

$T_1 = 77 + 70 + 69 = 216, \ldots, T_4 = 54 + 40 + 55 = 149,$

$B_1 = 77 + 85 + 60 = 222, \ldots, B_4 = 72 + 63 + 55 = 190,$

$Q_i = T_i - \sum_{j(i)} B_j/k$

Treat. Total $T_1 = 77 + 70 + 69 = 216$, etc., Block Total $B_1 = 77 + 85 + 60 = 222$

Total of blocks in which treat. i occurs $\sum_{j(i)} B_j = 222 + 191 + 171 = 584$, etc.

Adj. treat. Total $(Q_i) = T_i - \sum_{j(i)} B_j/k = 216 – 584/3 = 21.33$, etc.

Total SS (TSS) = Σ (observation)^2 – CF = $77^2 + 85^2 + \dots + 55^2$ – CF

$= 51442 – 49923 = 1519$

Treatment SS unadj. $(SST_U) = [\Sigma (T_i)^2]/r$ – CF = $153258 – 49923 = 1163$

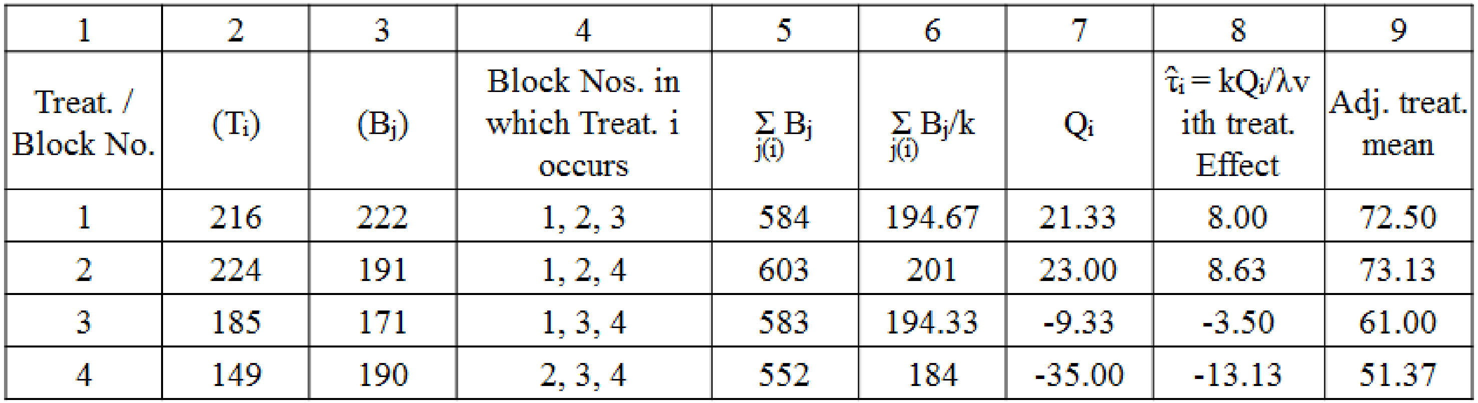
Block SS unadj. $(SSB_U) = [\Sigma(B_j)^2]/k$ – CF = $151106/3 – 49923 = 445.67$

Treatment SS adj. $(SST_A) = \Sigma \hat{\tau}_i Q_i = 861.34$

Error SS (SSE) = TSS – SSB_U – SST_A = $1519 – 445.67 – 861.34 = 211.99$

Block SS adj. $(SSB_A) = SST_A + SSB_U – SST_U = 861.34 + 445.67 – 1163 = 144.01$

Refer ANOVA table in next slide -



1	2	3	4	5	6	7	8	9
Treat. / Block No.	(T_i)	(B_j)	Block Nos. in which Treat. i occurs	$\sum_{j(i)} B_j$	$\sum_{j(i)} B_j/k$	Q_i	$\hat{\tau}_i = kQ_i/\lambda v$ ith treat. Effect	Adj. treat. mean
1	216	222	1, 2, 3	584	194.67	21.33	8.00	72.50
2	224	191	1, 2, 4	603	201	23.00	8.63	73.13
3	185	171	1, 3, 4	583	194.33	-9.33	-3.50	61.00
4	149	190	2, 3, 4	552	184	-35.00	-13.13	51.37

Treat. Total $T_1 = 77 + 70 + 69 = 216$, etc., Block Total $B_1 = 77 + 85 + 60 = 222$

Total of blocks in which treat. i occurs $\sum_{j(i)} B_j = 222 + 191 + 171 = 584$, etc.

Adj. treat. Total $(Q_i) = T_i - \sum_{j(i)} B_j/k = 216 - 584/3 = 21.33$, etc.

Total SS (TSS) = Σ (observation)^2 – CF = $77^2 + 85^2 + \ldots + 55^2$ – CF

= 51442 – 49923 = 1519

Treatment SS unadj. (SST_U) = $[\Sigma (T_i)^2]/r$ – CF = 153258 – 49923 = 1163

Block SS unadj. (SSB_U) = $[\Sigma(B_j)^2]/k$ – CF = 151106/3 – 49923 = 445.67

Treatment SS adj. (SST_A) = $\Sigma\, \hat{\tau}_i\, Q_i$ = 861.34

Error SS (SSE) = TSS – SSB_U – SST_A = 1519 – 445.67 – 861.34 = 211.99

Block SS adj. (SSB_A) = SST_A + SSB_U – SST_U = 861.34 + 445.67 – 1163 = 144.01

Refer ANOVA table in next slide -

ANOVA Table					
Source of Variation	DF	SS	MS	$F_{calculated}$	$F_{critical}$
SSBlocks (unadj.)	3	445.67			
SSTreatments (adj.)	3	861.34	287.11	6.77	5.41
SSBlocks (adj.)	3	144.01	48.00	1.13	5.41
SSTreatments (unadj.)	3	1163.00			
SSError	11 - 6 = 5	211.99	42.40		
SSTotal	12 – 1 = 11	1519.00			

Since for Treatments $F_{calculated}$ (6.77) > $F_{critical}$ (5.41), Treatment effects are significant.

Ex : Analyze the following BIBD

Balanced Incomplete Block Design for Catalyst Experiment

Treatment (Catalyst)	Block (Batch of Raw Material)			
	1	2	3	4
1	73	74	-	71
2	-	75	67	72
3	73	75	68	-
4	75	-	72	75

B1	B2	B3	B4
1	1	2	1
3	2	3	2
4	3	4	4

H_{0t} : All treatment effects are equal –vs- H_{at} : All treatment effects are not equal

H_{0b} : All block effects are equal –vs- H_{ab} : All block effects are not equal

Balanced Incomplete Block Design for Catalyst Experiment

Treatment (Catalyst)	Block (Batch of Raw Material)				
	1	2	3	4	$y_{i.}$
1	73	74	-	71	218
2	-	75	67	72	214
3	73	75	68	-	216
4	75	-	72	75	222
$y_{.j}$	221	224	207	218	$870 = y_{..}$

This is a BIBD with $v = a = 4$, $b = 4$, $k = 3$, $r = 3$, $\lambda = 2$ and $N = 12$. The analysis of this data is as follows. The Total Sum of Squares is –

$$SST = \Sigma\,(\text{obsn})^2 - G^2/N = \sum_i \sum_j (y_{ij})^2 - (y_{..})^2/12$$

$$= (73^2 + 74^2 + \ldots + 75^2) - 870^2/12 = 63156 - 63075 = 81$$

$$SSBlocks = 1/3[\sum_{j=1}^{4}(y_{.j})^2] - G^2/N = 1/3[221^2 + 224^2 + 207^2 + 218^2] - 870^2/12$$

$$= 63125 - 63070 = 55$$

$$Q_1 = T_1 - 1/3(B_1 + B_2 + B_4) = 218 - 1/3(221 + 224 + 218) = -9/3 = -3$$

$$Q_2 = T_2 - 1/3(B_2 + B_3 + B_4) = 214 - 1/3(224 + 207 + 218) = -7/3$$

$$Q_3 = T_3 - 1/3(B_1 + B_2 + B_3) = 216 - 1/3(221 + 224 + 207) = -4/3$$

$$Q_4 = T_4 - 1/3(B_1 + B_3 + B_4) = 222 - 1/3(221 + 207 + 218) = 20/3$$

$$\text{SSTreatment (adjusted)} = \text{SSTreat}_{.(\text{adj})} = [k\sum_{i=1}^{4}(Q_i)^2]/\lambda a = \Sigma\,\hat{\tau}_i\,Q_i$$

$$= 3[9\text{-}9/3)2 + (-7/3)2 + (-4/3)2 + (20/3)2]/(2\text{x}4) = 22.75$$

The error sum of squares is obtained by subtraction as -

$$\text{SSE} = \text{SST} - \text{SSTreat}_{.(\text{adj})} - \text{SSBlocks} = 81 - 22.75 - 55 = 3.25$$

$$Q'_1 = B_1\ \text{-}1/3(T_1 + T_3 + T_4) = 221 - 1/3(218 + 216 + 222) = 7/3$$

$$Q'_2 = B_2\ \text{-}1/3(T_1 + T_2 + T_3) = 224 - 1/3(218 + 214 + 216) = 24/3$$

$$Q'_3 = B_3\ \text{-}1/3(T_2 + T_3 + T_4) = 207 - 1/3(214 + 216 + 222) = -31/3$$

$$Q'_4 = B_4\ \text{-}1/3(T_1 + T_2 + T_4) = 218 - 1/3(218 + 214 + 222) = 0$$

$SSBlocks_{(adj.)} = [r\Sigma (Q'_i)^2]/\lambda b = 3[(7/3)^2 + (24/3)^2 + (-31/3)^2 + (0)^2]/(2x4) = 66.08$

$SSTreat. = \Sigma (T_i)^2/r – G^2/N = [(218)^2 + (214)^2 + (216)^2 + (222)^2]/3 – (870)^2/12$

$= 11.67$

ANOVA including both Treatments & Blocks

Source of Variation	DF	SS	MS	$F_{calculated}$	$F_{critical}$
SSTreatments (adj.)	3	22.75	22.75/3 = 7.58	7.58/0.65 = 11.66	5.41
SSTreatments (unadj.)	3	11.67			
SSBlocks (unadj.)	3	55.00			
SSBlocks (adj.)	3	66.08	66.08/3 = 22.03	22.03/0.65 = 33.89	5.41
SSError	11 - 6 = 5	3.25	3.25/5 = 0.65		
SSTotal	11	81.00			

Value of $F_{3,5}$ for 5% LOS, obtained from F-table, is 5.41 ie. $F_{critical} = 5.41$

Since both Treatments & Blocks $F_{calculated}$ (11.66) & (33.89) > $F_{critical}$ (5.41), Reject both H_{0t} and H_{0b}

Advantages and Disadvantages of BIBDs

Designs are complex and are looked up in tables.

Advantages

Small blocks are more homogeneous than large blocks, so experimental error is lower.

Use when there is variability within larger blocks.

Use to increase precision.

Disadvantages

Design can require a fixed number of treatments, a fixed number of reps or both.

More reps are needed.

The complexity of the analysis is increased.

There is unequal precision for certain comparisons of treatment means.

Uses

Use to reduce block size in single factor experiments when the number of treatments is large.

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

BSDCHZC355
Statistical Inference & Applications

Shaibal Kr. Sen
Session 10

innovate achieve lead

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

STATISTICAL INFERENCES

&

APPLICATIONS

Scheduled Topics to be covered

Design of Experiments : Incomplete Block Designs contd.

Simple Lattice Design

Youden Square Design

Simple Lattice Design (Quasi Factorial Design)

Lattice designs form an important class of useful Incomplete Block Designs.

Balanced Incomplete Block Designs (BIBD) are the most efficient designs in the class of Binary Incomplete Block Designs but these designs require usually a large number of replications and are not available for all combinations of parametric values. To overcome these difficulties Yates (1936) introduced a class of designs known as Quasi Factorial or Lattice Designs. The characteristic features of these designs are that the number of treatments is a perfect square and the block size is square root of this number. Moreover, incomplete blocks are combined in groups to form separate replications. The

Simple Lattice Design

number of replications of the treatments are flexible in these designs and are useful for situations in which a large number of treatments are to be tested. If the design has two replications in the treatments, it is called a simple lattice; if it has 3 replications it is called a triple-lattice and so on. In general, if the number of replications is m, it is called an m-ple lattice. Square Lattice designs can be constructed as depicted in next slide.

Statistical Analysis

Similar to BIBD, consider the model : $y_{ij} = \mu + \tau_i + \beta_j + e_{ij}$

Observation = General Mean + Treatment Effect + Block Effect + Random Error.

Random errors are assumed to be independently and identically distributed normally with mean zero and constant variance σ^2.

Now we compute -

G = Grand Total of observations.

$\bar{y}$ = Grand Mean = G/n, where $n = ms^2 = 2s^2$, where, s = block size

T_i = Sum of observations for treatment i, ($i = 1, 2, \ldots, s^2$)

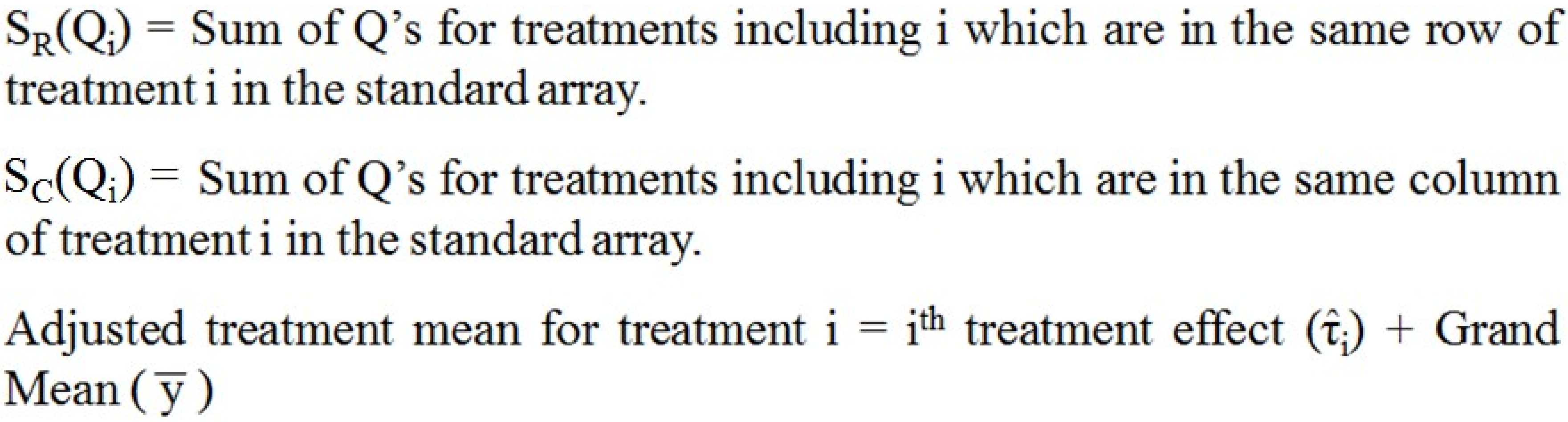
B_j = Sum of observations in block j, ($j = 1, 2, \ldots, ms$), where m = 2

CF = G^2/n, where $n = ms^2 = 2s^2$

Q_i = Adjusted $i^{\underline{th}}$ treatment total

= T_i – (Sum of block totals in which treatment i occurs)/Block size (s)

Statistical Analysis



$S_R(Q_i)$ = Sum of Q's for treatments including i which are in the same row of treatment i in the standard array.

$S_C(Q_i)$ = Sum of Q's for treatments including i which are in the same column of treatment i in the standard array.

Adjusted treatment mean for treatment i = i^{th} treatment effect ($\hat{\tau}_i$) + Grand Mean ($\overline{y}$)

Various Sums of Squares can be obtained as follows –

(i) Total Sum of Squares (TSS) = $\Sigma(\text{observations})^2 - CF$

(ii) Treatments Sum of Squares unadjusted (SST_U)

$$= \Sigma(T_i)^2/m - CF$$

(iii) Blocks Sum of Squares unadjusted (SSB_U) = $\Sigma(B_j)^2/s - CF$

(iv) Treatments Sum of Squares adjusted (SST_A) = $\Sigma t_i\, Q_i$

(v) Error Sum of Squares (SSE) = $TSS - SSB_U - SST_A$

(vi) Blocks Sum of Squares adjusted (SSB_A)

$$= SST_A + SSB_U - SST_U$$

The Analysis of variance for an m-ple lattice with $v = s^2$ treatments in blocks of size ‘s’ is given below -

ANOVA including both Treatments & Blocks

Source of Variation	DF	SS	MS	$F_{calculated}$
SSBlocks (unadj.)	$ms - 1$	SSB_U		
SSTreatments (adj.)	$s^2 - 1$	SST_A	MST	MST/MSE
SSBlocks (adj.)	$ms – 1$	SSB_A	MSB	MSB/MSE
SSError	$(s - 1)(ms - s - 1)$	SSE	MSE	
SSTotal	$ms^2 – 1$	TSS		

Ex : The following table gives the synthetic yields per plot of an experiment conducted with $3^2 = 9$ treatments using a simple lattice design.

	Replication 1				Replication 2		
Blocks	Treatments (yield per plot)			Blocks	Treatments (yield per plot)		
1	1 (8)	7 (5)	4 (3)	4	8 (2)	7 (2)	9 (7)
2	3 (3)	6 (2)	9 (6)	5	4 (3)	5 (3)	6 (3)
3	8 (3)	5 (7)	2 (3)	6	2 (2)	3 (4)	1 (6)

Analyze the data.

Analysis :

Grand Total, $G = 8 + 5 + \ldots + 6 = 72$

No. of observations, $n = 18$

Grand Mean, $\bar{y} = G/n = 72/18 = 4$, No. of replications = 2

Block size, $k = 3$, Correction Factor (CF) $= G^2/n = 72^2/18 = 288$

Note : $T_1 = 8 + 6 = 14$, etc.

$B_1 = 8 + 5 + 3 = 16$

Total of Block, in which treatment 1 occurs,

$$\sum_{j(1)} B_j = (8 + 5 + 3) + (2 + 4 + 6) = 16 + 12 = 28, \text{ etc.}$$

Adjusted Treatment Total, $Q_1 = T_1 - \sum_{j(1)} B_j/k = 14 - 28/3 = 4.67$

$$\hat{\tau}_1 = Q_1/2 + [S_R(Q_1) + S_C(Q_1)]/(2x3) = 4.67/2 + 1.001/6 = 2.50$$

Specimen examples : $T_1 = 8 + 6 = 14, \ldots, T_5 = 7 + 3 = 10, \ldots, T_9 = 6 + 7 = 13$

$B_1 = 8 + 5 + 3 = 16, \ldots, B_4 = 2 + 2 + 7 = 11, \ldots, B_6 = 2 + 4 + 6 = 12$

$\sum_{j(1)} B_j = (8 + 5 + 3) + (2 + 4 + 6) = 16 + 12 = 28$, (Total of Block, in which treatment 1 occurs)…,

$\sum_{j(5)} B_j = (8 + 5 + 3) + (2 + 4 + 6) = 16 + 12 = 28$ etc.

$\hat{\tau}_1 = Q_1/2 + [S_R(Q_1) + S_C(Q_1)]/(2x3) = 4.67/2 + 1.001/6 = 2.50$

Adjusted treatment mean for treatment i = i^{th} treatment effect (τ_i) + Grand Mean ($\bar{y}$)

Grand Mean, $\bar{y} = G/n = 72/18 = 4$

(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)
Treat/ Block No.	T_i	B_j	Block Nos. in which treat i occurs	$\sum_{j(i)} B_j$	$\sum_{j(i)} B_j/k$	Q_i (2) – (6)	$\hat{\tau}_i$	Adj. treat. Mean
1	14	16	1, 6	28	9.33	4.66	2.50	6.50
2	5	11	3, 6	25	8.33	-3.33	-2.16	1.84
3	7	13	2, 6	23	7.66	-0.66	0.33	4.33
4	6	11	1, 5	25	8.33	-2.66	-1.33	2.66
5	10	9	3, 5	22	7.33	2.66	0.50	4.50
6	5	12	2, 5	20	6.66	-1.66	-0.50	3.50
7	7		1, 4	27	9.00	-2.00	-0.83	3.16
8	5		3, 4	24	8.00	-3.00	-2.00	2.00
9	13		2, 4	22	7.33	5.66	3.50	7.50

Total Sum of Squares (TSS) $= \Sigma\,(\text{observation})^2 - CF$

$= 8^2 + 5^2 + \ldots. + 6^2 - 72^2/18 = 66$

Treatment Sum of Squares unadjusted (SST_U) $= [\Sigma\,(T_i)^2]/m - CF$

$= (14^2 + \ldots +13^2)/2 - 72^2/18 = 49$

Block Sum of Squares unadjusted (SSB_U) $= [\Sigma\,(B_j)^2]/s - CF$

$= (16^2 + \ldots + 12^2)/3 - 72^2/18 = 9.33$

Treatment Sum of Squares adjusted (SST_A) $= \Sigma\,\hat{\tau}_i Q_i = 51.44$

Block Sum of Squares adjusted (SSB_A) $= SST_A + SSB_U - SST_U$

$= 51.44 + 9.33 - 49 = 11.77$

Error Sum of Squares (SSE) $= TSS - SSB_U - SST_A$

$= 66 - 51.44 - 9.33 = 5.23$

ANOVA Table is shown in the next slide.

ANOVA Table

Source of Variation	DF	SS	MS	Fcalculated	Fcritical
SSBlocks (unadj.)	5	9.33			
SSTreatments (adj.)	8	51.44	6.43	4.91	6.04
SSBlocks (adj.)	5	11.77	2.35	1.79	6.26
SSTreatments (unadj.)	8	49.00			
SSError	17 - (5 + 8) = 4	5.23	1.31		
SSTotal	17				

$F_{0.05,8,4}$ and $F_{0.05,5,4}$ values obtained from F-table are 6.04 & 6.26 respectively.

As $F_{calculated}$ values < $F_{critical}$ value we conclude that Treatment effects and block effects are not significantly different.

Advantages and Disadvantages of Lattice Designs

The main advantage of Balanced Lattices is that a large number of treatments may be compared within relatively small blocks. Another advantage of balanced lattice designs is that each pair of treatments is compared with the same degree of precision because each treatment occurs together in the same block with every other treatment an equal number of times (usually once). Hence, to obtain a balanced lattice, some restrictions on the number of treatments and the number of blocks in the design are required. Consequently, balanced lattices are not available for 36, 100 and 144 treatments. The disadvantages of the design are the limitations for the number of allowable treatments, block sizes and replication. The analysis also becomes more complex and the designs are more difficult to construct as the number of treatments increase.

Youden Square Design

A Youden square design (YSD) is a design with incomplete columns by means of which two sources of variation can be eliminated. The rows of YSD form a Randomised Block Design (RBD) and the columns form a Balanced Incomplete Block Design (BIBD). These are basically symmetrical BIBD by which the block to block variation can be eliminated. The k units in each block be thought of occupying k different positions. With the help of YSD, the effects of such positions can be eliminated.

Youden square is a reduction from Randomized Complete Blocks to a Balanced Incomplete Design in which one row or column has been removed from a Latin square (so it is no longer square). It is a useful design for balancing out the effects of treatment order in a repeated-measures sequence.

Youden Square Design

Definition **:** A YSD is an arrangement of v treatments in a k x v rectangular array such that every symbol occurs exactly once in each row and the columns form a symmetrical Balanced Incomplete Block Design (BIBD) with parameters v = b, r = k, λ. Here, k = no. of rows, v = no. of columns, b = no. of treatment, r = no. of replicate and λ = no. of pair replicate.

YSD with 7 treatments arranged in 3 rows and 7 columns

1	2	3	4	5	6	7
2	3	4	5	6	7	1
4	5	6	7	1	2	3

Model of YSD

$y_{ij(m)} = \mu + \rho_i + c_j + \tau_m + e_{ij(m)}$ .... (a)

Where μ is the general mean, ρ_i is the $i^{\underline{th}}$ row effect, c_j is the $j^{\underline{th}}$ column effect, τ_m is the $m^{\underline{th}}$ treatment effect and $e_{ij(m)}$ are random errors assumed to be independently normally distributed. The model of response for a YSD is the same as in (a) with i = 1, 2, …, k; j = 1, 2, …, v and m = 1, 2, …, v.

Let $R_1, R_2, \ldots, R_k$ be the k row totals; $C_1, C_2, \ldots, C_v$ be the v column totals, $T_1, T_2, \ldots, T_v$ be the v treatment totals and G be the grand total.

Further let Q_m be adjusted $m^{\underline{th}}$ treatment total obtained by subtracting the sum of column means in which $m^{\underline{th}}$ treatment occurs from T_m (m = 1. 2. …, v). Further let $Q = (Q_1, Q_2, \ldots, Q_m)'$.

$\hat{\tau}_m = (k/\lambda v)Q_m,\ m = 1, 2, \ldots, v$ where, $Q_m = T_m - \sum_{j(m)} C_j\,/k$

H_{0R} : All Row effects are equal or All Row Means are equal

H_{0C} : All Column effects are equal or All Column Means are equal

H_{0T} : All Treatment effects are equal or All Treatment Means are equal

Analyze the following YSD. The yields are given below with 4 treatments.

	C_1	C_2	C_3	C_4
R_1	2 (1)	3 (2)	1 (3)	4 (2)
R_2	3 (2)	1 (3)	4 (5)	2 (4)
R_3	4 (4)	2 (1)	3 (2)	1 (5)

Here, k = no. of rows = 3, v = b = no. of columns = 4 = no. of treatments, no. of replication = 4, n = 12.

$R_1 = 1 + 2 + 3 + 2 = 8$; $R_2 = 2 + 3 + 5 + 4 = 14$; $R_3 = 4 + 1 + 2 + 5 = 12$

$C_1 = 1 + 2 + 4 = 7$; $C_2 = 2 + 3 + 1 = 6$; $C_3 = 3 + 5 + 2 = 10$; $C_4 = 2 + 4 + 5 = 11$

$T_1 = 3 + 3 + 5 = 11$; $T_2 = 1 + 1 + 4 = 6$; $T_3 = 2 + 2 + 2 = 6$; $T_4 = 4 + 5 + 2 = 11$

$G = 8 + 14 + 12 = 34$; $CF = G^2/n = 34^2/12 = 96.33$

$RSS = \Sigma\, y^2 = 1^2 + 2^2 + 3^2 + 2^2 + 2^2 + 3^2 + 5^2 + 4^2 + 4^2 + 1^2 + 2^2 + 5^2$

$= 18 + 54 + 46 = 118$

$Q_1 = T_1 – (C_2 + C_3 + C_4)/3 = 11 – (6 + 10 + 11)/3 = 2$

$Q_2 = T_2 – (C_1 + C_2 + C_4)/3 = 6 – (7 + 6 + 11)/3 = -2$

$Q_3 = T_3 – (C_2 + C_1 + C_3)/3 = 6 – (6 + 7 + 10)/3 = -1.66$

$Q_4 = T_4 – (C_4 + C_3 + C_1)/3 = 11 – (11 + 10 + 7)/3 = 1.66$

$TSS = \Sigma\, y^2 – CF = 118 – 96.33 = 21.67$

$SSR = [\Sigma(R_i)^2]/v – CF = (8^2 + 14^2 + 12^2)/4 – 96.33 = 101 – 96.33 = 4.67$

$SSC = [\Sigma(C_i)^2]/k – CF = (7^2 + 6^2 + 10^2 + 11^2)/3 – 96.33 = 102 – 96.33 = 5.67$

$SST = [k\, \Sigma(Q_m)^2]/\lambda v = [3\{(2)^2 + (-2)^2 + (-1.66)^2 + (1.66)^2\}]/3x4 = 3.36$

$SSE = TSS – SSR – SSC – SST = 21.67 – 4.67 – 5.67 – 3.36 = 7.97$

ANOVA Table

Source of Variation	DF	SS	MS	$F_{calculated}$	$F_{critical}$
SSRows	k - 1 = 2	4.67	4.67/2 = 2.335	2.335/2.66 = 0.88	9.55
SSColumns	v - 1 = 3	5.67	5.67/3 = 1.89	1.89/2.66 = 0.71	9.28
SSTreatments (eliminating rows & columns)	v - 1 = 3	3.36	3.36/3 = 1.12	1.12/2.66 = 0.42	9.28
SSError	vk - 2v - k + 2 = 12 - 8 - 3 + 2 = 3	7.97	7.97/3 = 2.66		
SSTotal	vk - 1 = 11	21.67			

k = no. of rows = 3, v = b = no. of columns = 4 = no. of treatments, no. of replication = 4, n = 12.

The significance of treatment effects can be tested by comparing the F-value of table for $F_{0.05,(v-1),(vk-2v-k+2)}$. The calculation of treatment SS is identical to the one for a BIB design and the other calculations like Rows SS, Columns SS are the routine ones.

Since the calculated F for all parameters less than Fcritical values, null hypothesis for H_{0R}, H_{0C} and H_{0T} are all acceptable.

Note **:** Error SS = Total SS – Rows SS – Columns SS -Treatments (eliminating rows & columns) SS = 21.67 – 4.67 – 5.67 – 3.36 = 7.97

Illustration : In one of the experiments, the experimenter is interested in making comparison among 7 treatments and there are 28 experimental units available. These 28 experimental units are arranged in a Youden Square Design with 4 rows and 7 columns with one observation per cell. The parameters of the design are v (number of treatments) = 7, p ← k (number of rows) = 4, q ← v (number of columns) = 7, r (replication of treatments) = 4. The layout of the design along with the observations is given below -

2(4.00)	3(5.30)	4(1.10)	5(16.90)	6(16.90)	7(10.30)	1(294.00)
7(17.50)	1(220.00)	2(12.20)	3(15.50)	4(11.00)	5(26.50)	6(27.20)
6(37.00)	7(26.00)	1(310.00)	2(22.70)	3(24.20)	4(21.40)	5(31.30)
5(46.80)	6(44.20)	7(34.30)	1(282.00)	2(33.70)	3(33.70)	4(30.50)

The analysis of variance for the above design for testing the equality of treatment effects is obtained in the next slide.

ANOVA Table

Source of Variation	DF	SS	MS	Fcalculated	Fcritical
Due to Rows	3	3319.0068	1106.336	4.89	3.49
Due to Columns (unadjusted)	6	27595.4086	4599.2348	20.31	3.00
Treatments (adjusted)	6	195027.72	32504.6207	143.56	3.00
Error	12	2717.0814	226.4235		
Total	27	228659.2211			

no. of rows = 4, no. of columns = 7, no. of treatments = 7, no. of replication = 4.

$F_{0.05,3,12} = 3.49$; $F_{0.05,6,12} = 3$

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

BSDCHZC355
Statistical Inference & Applications

Shaibal Kr. Sen

Session 11

innovate achieve lead

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

STATISTICAL INFERENCES

&

APPLICATIONS

Scheduled Topics to be covered

Factorial Experiments -

Basic Definitions and principles

Interactions and Analysis of -

- 2^2 Factorial experiments
- 2^3 Factorial experiments

Factorial Designs

Main Effects and Interaction Effects Simultaneously:

To study the significant difference between treatment (or levels of factor) effects (main effects).

To study the significant difference between interactive treatment effects.

Experiments where the effects of more than one factor, say variety, manure etc. are considered together are called factorial experiments, while experiments with one factor, say only variety or manure, may be called simple experiments. Consider a simple case If there are p different varieties, then we shall say that there are p levels of the factor ‘variety’. Similarly, the second factor ‘manure’ may have q levels, ie. there may be q different manures or different doses of the same manure. Then this factorial design will be called a pxq experiment. As a different example, the two factors may be two different manures, say nitrogen and phosphate and at different doses, and p and q different doses, respectively. Then this will also give a pxq experiment. We shall consider only the simplest cases, viz. cases of a n factors each at 2 levels or what are known as 2^n experiments, where n is any positive integer greater than or equal to 2.

Let us examine a two-factor factorial so that we can set up some basic notation as well as introduce some fundamental ideas. Notationally the designs are 2^K , where K = number of factors, and 2^K = number of treatment combination runs (usually just called "treatments").

For example if K = 2, the design requires $2^2 = 4$ runs.

A 2^2 experiment **:** Let us consider two factors – A and B, each at 2 levels. Following Yates, we denote by a or b one of two levels at which the corresponding factor (denoted by capital letter) occurs and for definiteness we shall call this the second level. The first level of A or B will be signified by the absence of the corresponding letter in the treatment combination. Now with 2 factors, each at 2 levels, there will be 2x2 = 4 treatment combinations. They are enumerated below –

(1) : A and B both at first levels or at lower level,
a : A at second level and B at first level or A at high level & B at low level,
b : A at first level and B at second level or A at low level & B at high level,
ab : A and B both at second levels or A at high level & B at high level,

These four treatment combinations may be compared using a CRD (Completely Randomized Design) or RBD (Randomized Block Design) or an LSD (Latin Square Design). For a 2^2 experiment in r blocks, the treatment combinations t = 4. And the analysis of a 2^2 experiment in a Latin square design m = 4. In a factorial experiment one is more interested in the separate tests about Main Effects and Interaction Effects, which are performed by splitting the treatment SS carrying 3 d.f. into 3 orthogonal components, each carrying a single d.f. and each associated either with a main effect or an interaction.

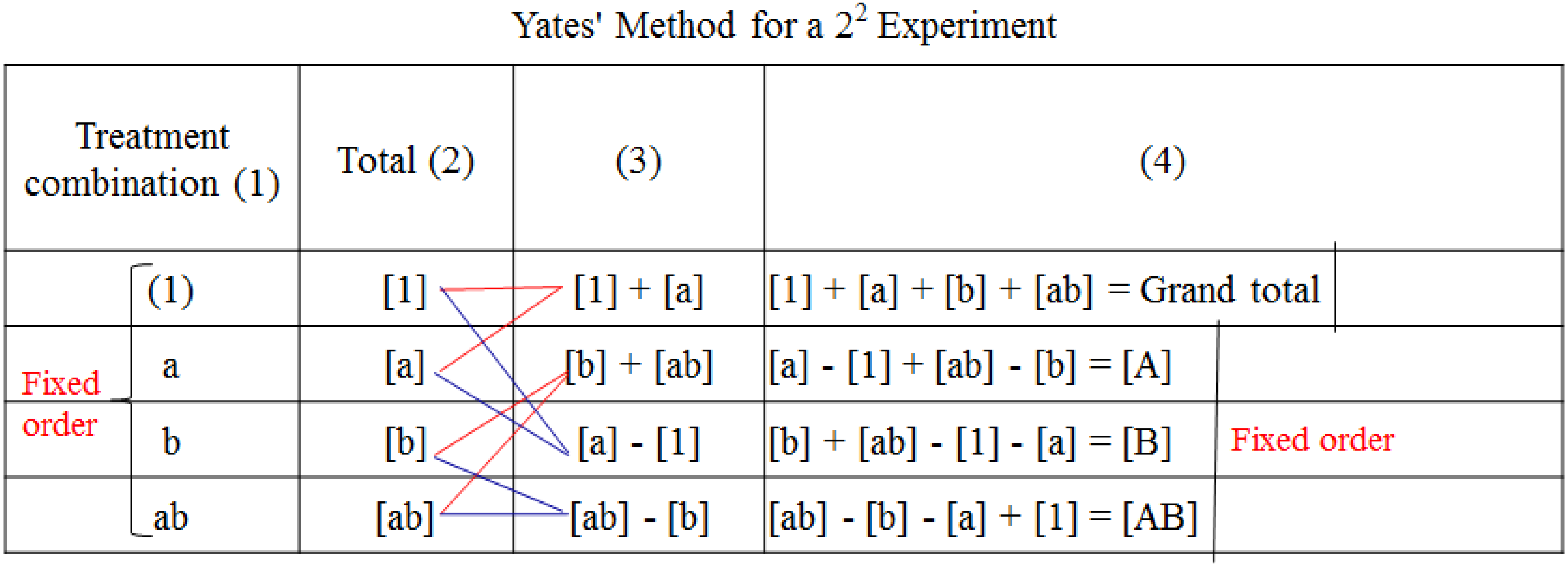
Main effect and Interaction effect : The symbols [a] and (a) will be used to denote the total and mean (respectively) of all observations receiving the treatment combination a. The letters A, B and AB, when they refer to numbers, will be used to stand for main effects due to factors A and B and AB due to the interaction of the two factors.

It is convenient to obtain the factorial effects and their SSs from the treatment totals rather than from the treatment means. We define the factorial effect totals as follows –

$$\left.\begin{aligned}{[A]} &= [ab]-[b]+[a]-[1]\\ {[B]} &= [ab]+[b]-[a]-[1]\\ {[AB]} &= [ab]-[b]-[a]+[1]\end{aligned}\right\} \quad \ldots\ldots\ldots\ldots(i)$$

Then the SS due to any main effect or the interaction effect is obtained by multiplying the square of the effect total with the reciprocal of 4r, where r is the common replication number. Thus

$$\left.\begin{aligned}&\text{SS due to main effect of A} &&= [A]^2/4r \text{ with } 1 \text{ d.f.}\\ &\text{SS due to main effect of B} &&= [B]^2/4r \text{ with } 1 \text{ d.f.}\\ &\text{SS due to interaction effect of AB} &&= [AB]^2/4r \text{ with } 1 \text{ d.f.}\end{aligned}\right\} \quad \ldots\ldots\ldots\ldots(ii)$$



Yates' Method for a 2^2 Experiment

Treatment combination (1)	Total (2)	(3)	(4)
(1)	[1]	[1] + [a]	[1] + [a] + [b] + [ab] = Grand total
a	[a]	[b] + [ab]	[a] - [1] + [ab] - [b] = [A]
b	[b]	[a] - [1]	[b] + [ab] - [1] - [a] = [B]
ab	[ab]	[ab] - [b]	[ab] - [b] - [a] + [1] = [AB]

Fixed order

Fixed order

It is then a simple matter to express the factorial effect totals or the SSs in terms of the factorial effects, main or interaction, remembering that a factorial effect total is 2r times the corresponding factorial effect. Thus the factorial effects are as follows –

$$\left.\begin{array}{l}\text{Main effect of } A = [A]/2r \\ \text{Main effect of } B = [B]/2r \\ \text{Interaction } AB \quad = [AB]/2r\end{array}\right\} \;\ldots\ldots\ldots\ldots(\text{iii})$$

and the SS due to a factorial effect is rx(factorial effect)^2.

The test for the significance of any factorial effect, main effect or interaction, may now be obtained by computing –

F = (MS due to factorial effect)/MSE

ANOVA Table for 2^2 experiment in r Randomised Blocks

Source of variation	d.f.	SS	MS	$F_{calculated}$
Blocks	r - 1	SS (Blocks)	MS (Blocks)	
Main effect A	1	$[A]^2/4r$	MSA	MSA/MSE
Main effect B	1	$[B]^2/4r$	MSB	MSB/MSE
Interaction AB	1	$[AB]^2/4r$	MS(AB)	MS(AB)/MSE
Error	3(r - 1)	By subtraction	MSE	
Total	4r - 1	$\Sigma(y_{ij} - y_{oo})^2$		

Ex : A 2^2 experiment : in six (Y = 6) randomized blocks was conducted in order to obtain an idea of the interaction : spacing (s) x number of seedlings per hole (N) along with the effects of different types of spacing and different numbers of seedlings per hole, while adopting the Japanese method of cultivation.

The levels of two factors are –

S : { 8” spacings in between
10” spacings in between

and

N : { 3 seedlings per hole
4 seedlings per hole

The field plan and yield of dry Aman paddy (in kg.) are given in next slide.

Block 1

(1)	s	ns	n	Ttl
117	106	109	114	446

Block 2

ns	(1)	s	n	Ttl
114	120	117	114	465

Block 3

(1)	n	s	ns	Ttl
111	117	114	106	448

Block 4

ns	n	s	(1)	
98	121	112	108	439

Block 5

ns	s	(1)	n	
75	97	73	38	283

Block 6

n	(1)	ns	s	
58	81	105	117	361

Analyse the data to find out if there are any significant treatment effects – main or interaction.

We apply Yates' method to find the total effects.

Yates' Method for a 2^2 Experiment

Treatment combination (1)	Total yield from all blocks (2)	(3)	(4)	Average effect
(1)	[1] = 610	[1] + [n] = 1172	[1] + [n] + [s] + [ns] = 2442 = Grand total (G)	
n	[n] = 562	[s] + [ns] = 1270	[n] - [1] + [ns] - s] = -104 = [N]	[N]/2x6
s	[s] = 663	[n] - [1] = -48	[s] + [ns] - [1] - [n] = 98 = [S]	[S]/2x6
ns	[ns] = 607	[ns] - [s] = -56	[ns] - [s] - [n] + [1] = -8 = [NS]	[NS]/2x6

Now perform the randomized block analysis –

Treatment totals are : [1] = 117 + 120 + 111 + 108 + 73 + 81 = 610

[n] = 114 + 114 + 117 + 121 + 38 + 58 = 562

[s] = 106 + 117 + 114 + 112 + 97 + 117 = 563 and

[ns] = 109 + 114 + 106 + 98 + 75 + 105 = 607.

Grand Total (G) = [1] + [n] + [s] + [ns] = 610 + 663 + 562 + 607 = 2442

Six block totals are : 446, 465, 448, 439, 283 and 361.

Raw total SS = $\Sigma y^2 = 117^2 + 106^2 + \ldots\ldots + 117^2 = 2{,}59{,}024$

Correction Factor (CF) = $G^2/N = (2442)^2/24 = 2{,}48{,}473.5$

Total SS (TSS) = RSS – CF = 2,59,024 – 2,48,473.5 = 10,550.5

Block SS (BSS) = $(446^2 + 465^2 + \ldots. + 361^2)/4$ – CF

= 2,54,744 – 2,48,473.5 = 6270.5

Treatment SS (TrSS) = $\{(610^2 + \ldots. + 607^2)/6\}$ – 2,48,473.5

= 1495962/6 - 2,48,473.5 = 853.5

Error SS = TSS – BSS – TrSS = 10,550.5 – 6,270.5 - 853.5 = 3,426.5

Also, SS due to N = $[N]^2/4r = (-104)^2/24 = 450.667$

SS due to S = $[S]^2/4r = (98)^2/24 = 400.167$

SS due to NS = $[NS]^2/4r = (-8)^2/24 = 2.67$

ANOVA Table is presented in next slide

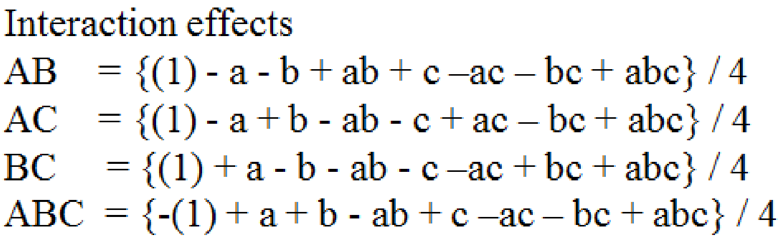
ANOVA Table for 2^2 experiment

Source of variation	d.f.	SS	MS	$F_{calculated}$	$F_{critical}$
Blocks	5	6,270.50	1254.1		$F_{0.05,1,15}$ = 4.54
N	1	450.667	450.667	1.973	
S	1	400.167	400.167	1.752	
NS	1	2.667	2.667	<1	
Error	23 - 8 = 15	3,426.50	228.433		
Total	23	10,550.50			

There is no significant main or interaction effects present in the above experiment, as in each of the cases the computed value of F is less than the corresponding theoretical value (critical value) at the 5% level.

A 2^3 experiment **:** We now consider the case of three factors – A, B and C, each at 2 levels, where a, b and c will denote the second levels of the factors, respectively, 2x2x2 = 8 treatment combinations written in the systematic order are : (1), a, b, ab, c, ac, bc, abc.

The 8 treatment combinations may be compared in any of the designs viz. – CRD, RBD or LSD. The analysis will be same as in the corresponding design, the number of treatments being t = 8 in CRD and RBD and m = 8 in LSD. The treatment SS has 7 d.f. We next divide it into 7 orthogonal contrasts of the 8 treatment means (or totals) with the help of the main effects and interactions. In a three-factor experiment there are three main effects – A, B, C; and three first order interactions – AB, AC, BC; and one second-order (or three-factor) interaction – ABC.



Interaction effects
AB = {(1) - a - b + ab + c –ac – bc + abc} / 4
AC = {(1) - a + b - ab - c + ac – bc + abc} / 4
BC = {(1) + a - b - ab - c –ac + bc + abc} / 4
ABC = {-(1) + a + b - ab + c –ac – bc + abc} / 4

SS due to factorial effects and tests of significance of factorial effects **:**

We define factorial effect totals by combining the 8 treatment totals with signs given in the above table. Thus

[A] = [abc] - [bc] + [ac] - [c] + [ab] - [b] + [a] - [1]

and similarly the other effect totals are obtained.

The SS due to factorial effect is obtained by multiplying the square of the corresponding effect total by the reciprocal of 8r, where r is common replication number. Thus, the test for the significance of any factorial effect, main effect or interaction may now be obtained by computing

F = MS due to factorial effect / MSE

where, MSE is the error MS of the analysis of variance table of the corresponding design. This F follows the F distribution with 1, 7(r – 1) d.f. Hence, the hypothesis of the absence of the factorial effect is rejected at the level α if for our data

$F > F_{\alpha,1,7(r-1)}$;

otherwise, the hypothesis is accepted. 7(r - 1) is the error d.f. for a 2^3 experiment conducted in r randomized blocks.

Run	Labels	A	B	C	AB	AC	BC	ABC
1	(1)	-	-	-	+	+	+	-
2	a	+	-	-	-	-	+	+
3	b	-	+	-	-	+	-	+
4	ab	+	+	-	+	-	-	-
5	c	-	-	+	+	-	-	+
6	ac	+	-	+	-	+	-	-
7	bc	-	+	+	-	-	+	-
8	abc	+	+	+	+	+	+	+

Interaction effects
AB = {(1) - a - b + ab + c –ac – bc + abc} / 4
AC = {(1) - a + b - ab - c + ac – bc + abc} / 4
BC = {(1) + a - b - ab - c –ac + bc + abc} / 4
ABC = {-(1) + a + b - ab + c –ac – bc + abc} / 4

Yates' method of computing factorial effect totals for a 2^3 experiment				
Treatment combination (1)	Total (2)	(3)	(4)	(5)
1)	[1]	[1] + [a]	[b] + [ab] + [1] + [a]	[bc] + [abc] + [c] + [ac] + [b] + [ab] + [1] + [a] = Grand total (G)
a	[a]	[b] + [ab]	[bc] + [abc] + [c] + [ac]	[abc] - [bc] + [ac] - [c] + [ab] - [b] + [a] - [1] = [A]
b	[b]	[c] + [ac]	[ab] - [b] + [a] - [1]	[bc] + [abc] - [c] - [ac] + [b] + [ab] - [1] - [a] = [B]
ab	[ab]	[bc] + [abc]	[abc] - [bc] + [ac] - [c]	[abc] - [bc] - [ac] + [c] + [ab] - [b] - [a] + [1] = [AB]
c	[c]	[a] - [1]	[b] + [ab] - [1] - [a]	[bc] + [abc] + [c] + [ac] - [b] - [ab] - [1] - [a] = [C]
ac	[ac]	[ab] - [b]	[bc] + [abc] - [c] - [ac]	[abc] - [bc] + [ac] - [c] - [ab] + [b] - [a] + [1] = [AC]
bc	[bc]	[ac] - [c]	[ab] - [b] - [a] + [1]	[bc] + [abc] - [c] - [ac] - [b] - [ab] + [1] + [a] = [BC]
abc	[abc]	[abc] - [bc]	[abc] - [bc] - [ac] + [c]	[abc] - [bc] - [ac] + [c] - [ab] + [b] + [a] - [1] = [ABC]

(+)

Half

(-)

ANOVA Table for a 2^3 experiment in r Randomised Blocks

Source of variation	d.f.	SS	MS	F
Blocks	r - 1	SS (Blocks)	MS (Blocks)	
Main effect A	1	$[A]^2/8r$	MSA	MSA/MSE
Main effect B	1	$[B]^2/8r$	MSB	MSB/MSE
Main effect C	1	$[C]^2/8r$	MSC	MSC/MSE
Two factor Interaction AB	1	$[AB]^2/8r$	MS(AB)	MS(AB)/MSE
Two factor Interaction AC	1	$[AC]^2/8r$	MS(AC)	MS(AC)/MSE
Two factor Interaction BC	1	$[BC]^2/8r$	MS(BC)	MS(BC)/MSE
Three factor Interaction ABC	1	$[ABC]^2/8r$	MS(ABC)	MS(ABC)/MSE
Error	7(r - 1)	SSE (by subtraction)	MSE	
Total	8r - 1	$\Sigma(y_{ij} - y_{oo})^2$		

Yates' method of computing factorial effect totals

Yates gives a systematic method of obtaining the various effect totals for any 2^n experiment without writing down the algebraic expressions. We shall describe it for the 2^2 experiment, but it can be easily extended to the case of any 2^n experiment.

The steps are as follows –
(i) First write down the 4 treatment combinations systematically in the 1st column, starting with the treatment combination (1) and then introducing the letters a, b in turn. After introducing a letter, write down its combination with all the previous treatment combinations and then introduce a new letter. Repeat it until all the letters (n letters in case of a 2^n experiment) have exhausted.

(ii) Next, write down the treatment total from all the replicates in the 2nd column against the appropriate treatment combination.

Yates' method of computing factorial effect totals

(iii) The first two columns we get from the observed data. For obtaining column 3, we break the even number of values in the 2nd column into consecutive pairs (1, 2; 3, 4; etc.) Then in the 1st half of the 3rd column we write down the sums of the values in these pairs in order and in the 2nd half of 3rd column we write down in order the differences of the values in the pairs in the 2nd column (the 1st member subtracted from 2nd member of a pair).

(iv) We next break the values in the 3rd column into consecutive pairs and put the sums and differences of the members of these pairs in order in the 4th column.

For a 2^2 experiment the 4th column values give the factorial effect totals corresponding to the treatment combinations occurring in the corresponding position of the 1st column.

Yates' method of computing factorial effect totals

For a 2^n experiment we are to repeat n times the operations of columns 3 and 4 and then the values in the $(n + 2)^{nd}$ column will be the factorial effect totals, the 1st entry in the last column being always the grand total.

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

BSDCHZC355
Statistical Inference & Applications

Shaibal Kr. Sen
Session 12

innovate achieve lead

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

STATISTICAL INFERENCES

&

APPLICATIONS

Scheduled Topics to be covered

3^2 Factorial experiments

Spit Plot Design

3^2 Design

When factors are taken at three levels instead of two, the scope of an experiment increases. It becomes more informative. A study to investigate if the change is linear or quadratic is possible when the factors are at three levels. The more the number of levels the better, yet the number of the levels of the factors cannot be increased too much as the size of the experiment as the size of the experiment increases too rapidly with them. Let us begin with two factors A and B, each at three levels, say 0, 1 and 2 (3^2 factorial experiment). The treatment combinations are –

00	a_0b_0	(1)	Factors A and B both at 1st levels
10	a_1b_0	a	Factors A is at 2nd level and B is at 1st level
20	a_2b_0	a^2	Factors A is at 3rd level and B is at 1st level
01	a_0b_1	b	Factors A is at 1st level and B is at 2nd level
11	a_1b_1	ab	Factors A and B both at 2nd level
21	a_2b_1	a^2b	Factors A is at 3rd level and B is at 2nd level
02	a_0b_2	b^2	Factors A is at 1st level and B is at 3rd level
12	a_1b_2	ab^2	Factors A is at 2nd level and B is at 3rd level
22	a_2b_2	a^2b^2	Factors A and B both at 3rd levels

3^2 Design

As learnt during previous session there will be 7 factorial effects – 3 main factor effects (A, B & C), 3 two factor Interaction effects (AB, AC & BC) and 1 three factor Interaction effect (ABC).

Any standard design can be adopted for the experiment. The main effects A, B can respectively be divided into linear and quadratic components each with 1 df as A_L, A_Q, B_L and B_Q. Accordingly AB can be partitioned into four components as A_LB_L, A_LB_Q, A_QB_L, A_QB_Q, each 1 df. The coefficients of the treatment combinations to obtain the above effects are given as –

Nine (3^2) Treatment Totals **:** [1] [a] [a^2] [b] [ab] [a^2b] [b^2] [ab^2] [a^2b^2]

Split-Plot Design.

In some multifactor factorial experiments, we may not be able to completely randomize the order of the runs. This often results in a generalization of the factorial design called a split-plot design.

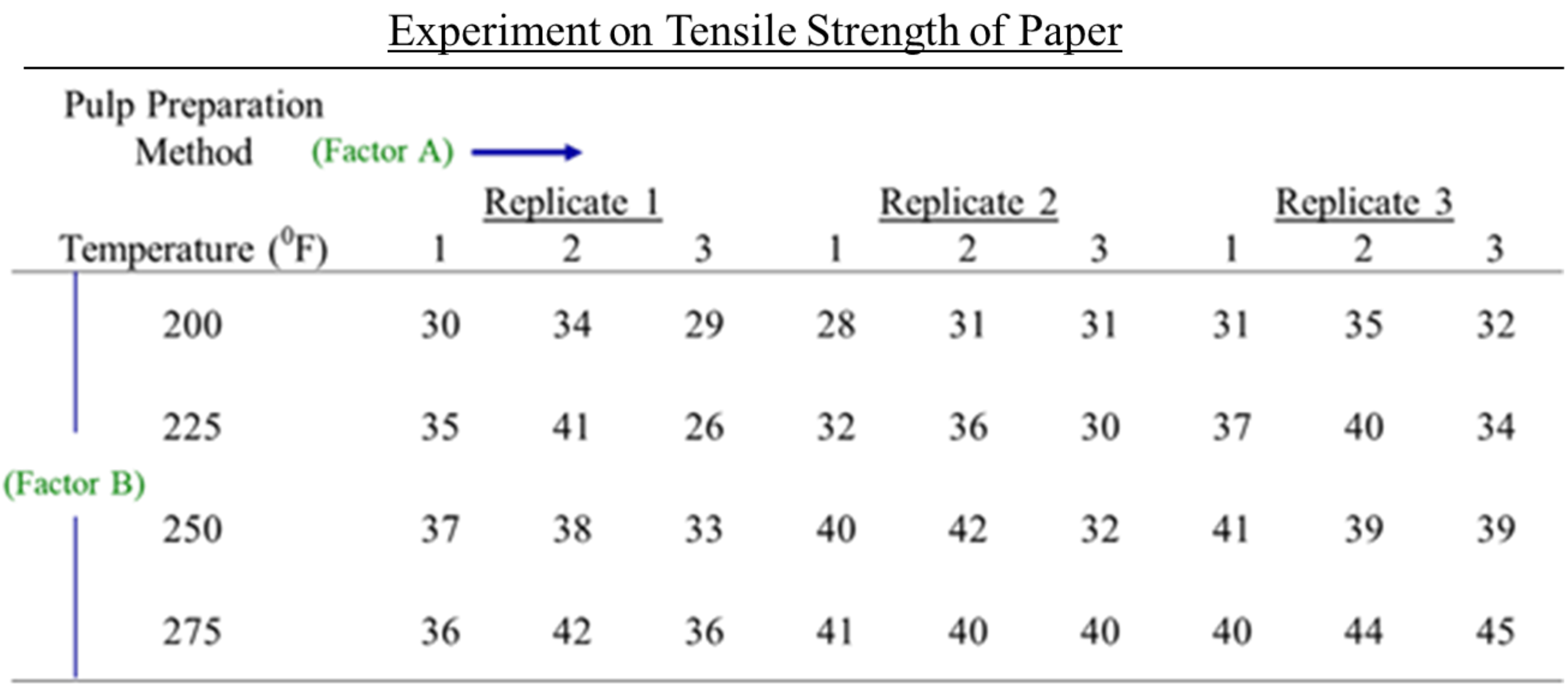
As an example, consider a paper manufacturer who is interested in three different pulp preparation methods. The methods differ –

(i) in the amount of hardwood in the three different pulp mixtures and

(ii) four different cooking temperatures for the pulp.

The experimenter wishes to study the effect of these two factors on the tensile strength of the paper. Each replicate of a factorial experiment requires 3 x 4 = 12 observations and the experimenter has decided to run three replicates. This will require total 36 runs. The experimenter decides to conduct the experiment as follows. A batch of pulp is produced by one of the three methods under study. Then this batch is divided into four samples and each sample is cooked at one of the four temperatures. Then a second batch of pulp is made up using another of the three methods. This second batch is also divided into four samples that are tested at the four temperatures. The process is then repeated until all three replicates (36 runs) are obtained. The data are shown in table above.

Initially, we might consider this to be factorial experiment with three levels of preparation method (factor A) and four levels of temperature (factor B). If this is the case, then the order of experimentation within each replicate should be completely randomized. That is, we should randomly select a treatment combination (a preparation method and a temperature) and obtain an observation, then we should randomly select another treatment combination and obtain a second observation, and so on, until all 36 observations have been taken. However, the experimenter did not collect the data this way. He made up a batch of pulp and obtained observations for all four temperatures from that batch. Because of the economics of preparing the batches and the size of the batches, this is the only feasible way to run this experiment. A completely randomized factorial experiment would require 36 batches of pulp, which is completely unrealistic. The split-plot design requires only 9 batches total. Obviously, the split-plot design has resulted in considerable experimental efficiency.



Experiment on Tensile Strength of Paper

Pulp Preparation
Method (Factor A) →

(Factor B)

	Replicate 1			Replicate 2			Replicate 3		
Temperature (^{0}F)	1	2	3	1	2	3	1	2	3
200	30	34	29	28	31	31	31	35	32
225	35	41	26	32	36	30	37	40	34
250	37	38	33	40	42	32	41	39	39
275	36	42	36	41	40	40	40	44	45

The design used in our example is a split-plot design. In this split-plot design we have 9 whole plots, and the preparation methods are called the whole plot or main treatments. Each whole plot is divided into four parts called subplots (or split-plots), and one temperature is assigned to each. Temperature is called the subplot treatment. Note that if other uncontrolled or undesigned factors are present and if these uncontrolled factors vary as the pulp preparation methods are changed, then any effect of the undesigned factors on the response will be completely compounded with the effect of the pulp preparation methods. Because the whole-plot treatments in a split-plot design are confounded with the whole-plots and the subplot treatments are not confounded, it is best to assign the factor we are most interested in to the subplots, if possible.

This example is fairly typical of how the split-plot design is used in an industrial setting. Notice that the two factors were essentially “applied” at different times. Consequently, a split-plot design can be viewed as two experiments “combined” or superimposed on each other. One “experiment” has the whole-plot factor applied to the large experimental units (or it is a factor whose levels are hard to change) and the other “experiment” has sub-plot factor applied to the smaller experimental units (or it is a factor whose levels are easy to change).

A good example of split-plot design is about a paper manufacturer who wants to analyze the effect of three pulp preparation methods and four cooking temperatures on the tensile strength of the paper. The experimenter wants to perform three replicates of this experiment on three different days each consisting of 12 runs (3 × 4). The important issue here is the fact that making the pulp by any of the methods is cumbersome. Thus method is a "hard to change" factor. It would be economical to randomly select any of the preparation methods, make the blend and divide it into four samples and cook each of them with one of the four cooking temperatures. Then the second method is used to prepare the pulp and so on. As we can see, in order to achieve this economy in the process, there is a restriction on the randomization of the experimental run.

In field experiments, sometimes a factor has to be applied to a large experimental unit. This is true when the different methods of ploughing or irrigation are to be compared. And in such cases it is possible to introduce a second factor, which does not require large plots, with a small number of levels into the same experiment, at a little extra cost. This is done by splitting the plots (called whole plots) of the first factor into as many sub-plots as there are levels of the second factor.

A split-plot design with an RBD for the first set of treatments (called “the whole-plot treatments’) is obtained by allotting the whole-plot treatments at random to the whole plots of a block and then randomizing the second set of treatments (“called the sub-plot treatments”) to the sub-plots within each whole plot.

Ex : A variety-manurial experiment was conducted by allotting the three varieties V_1, V_2 and V_3 at random to the plots of four randomized blocks and then, splitting each plot into four sub-plots, the four manures M_1, M_2, M_3 and M_4 were applied at random within each plot. The plan and yield are shown below. Analyse the data to find out if there are any effects due to manure or variety of interaction between variety and manure.

Variety	BLOCK			
	I	II	III	IV
V_1	609	450	488	545
V_2	920	870	833	1118
V_3	1067	1072	1093	905

p = 3 = number of varieties (A)

q = 4 = number of manures (B)

r = 4 = number of replicates or blocks

N = pqr = 3x4x4 = 48

Block I	V_1	V_3	V_2
	M_1 94	M_4 440	M_2 250
	M_3 220	M_2 297	M_1 147
	M_2 185	M_3 218	M_3 248
	M_4 110	M_1 112	M_4 275
Total	609	1067	920

Block II	V_2	V_1	V_3
	M_1 135	M2 160	M_4 370
	M_4 290	M4 95	M_1 140
	M_3 180	M3 124	M_3 340
	M_2 265	M1 71	M_2 222
	870	450	1072

Block III	V_1	V_2	V_3
	M_1 78	M_3 196	M_2 235
	M_3 135	M_4 262	M_3 260
	M_4 130	M_1 155	M_1 115
	M_2 145	M_2 220	M_4 483
Total	488	833	1093

Block IV	V_1	V_3	V_2
	M_1 81	M2 246	M_3 296
	M_2 175	M3 191	M_2 260
	M_4 175	M1 145	M_1 112
	M_3 114	M4 323	M_4 450
	545	905	1118

$M_1V_1 = 94 + 71 + 78 + 81 = 324$; $M_1V_3 = 112 + 140 + 115 + 145 = 512$

$M_2V_3 = 297 + 222 + 235 + 246 = 1000$; $M_3V_3 = 218 + 340 + 260 + 191 = 1009$

Soln : We draw up the block-variety table for obtaining the whole-plot analysis

Variety	BLOCK				Total
	I	II	III	IV	
V_1	609	450	488	545	2092
V_2	920	870	833	1118	3741
V_3	1067	1072	1093	905	4137
Total	2596	2392	2414	2568	9970

$CF = G^2/N = 9970^2/pqr = 9970^2/3x4x4 = 2{,}070{,}852.08333$

$\text{Block SS} = (2596^2 + 2392^2 + 2414^2 + 2568^2)/pq - CF$

$= 24882900/12 - 2{,}070{,}852.0833 = 2{,}073{,}575 - 2{,}070{,}852.0833$
$= 2{,}722.91667$

$\text{Variety SS} = (2092^2 + 3741^2 + 4137^2)/rq - CF$

$= 35486314/16 - 2{,}070{,}852.0833 = 2{,}217{,}894.625 - 2{,}070{,}852.0833$
$= 147{,}042.5417$

Error (I) SS = $(609^2 + 920^2 + \dots + 1118^2 + 905^2)/4 – 9970^2/48$ – Variety SS - Block SS

8957010/4 – 2,070,852.08333 – 147,042.54167 – 2,722.91667 = 18,6,34.9583

Now draw up the Variety-Manure table to obtain the Manure SS and Interaction SS

Variety → Manure ↓	V_1	V_2	V_3	Total
M_1	324	559	512	1395
M_2	665	900	1000	2565
M_3	593	1005	1009	2607
M_4	510	1277	1616	3403
Total	2092	3741	4137	9970

Manure SS = $(1395^2 + 2565^2 + 2607^2 + 3403^2)/rp – 9970^2/48$

= 26902108/12 – 2,070,852.08333 = 170,990.25

Variety x Manure SS = $(324^2 + 665^2 + \dots 1009^2 + 1616^2)/r - 9970^2/48$

– Manure SS – Variety SS

= 9813866/4 – 2,388,884.875 =64,581.625

Raw Total SS = Σy^2 = Sum of squares of 48 values from Variety-Manure

square = $94^2 + 440^2 + \dots + 323^2 + 450^2$ = 2,500,068

Total SS = RSS – CF = 2,500,068 - 2,070,852.0833 = 429,215.91667

Error (II) SS = Total SS – Variety SS – Var x Man SS – Man SS – Block SS

= 25,243.62493 (By subtraction)

ANOVA of the Split-Plot Design

Source of variation	df	SS	MS = SS/df	Fcalculated
Blocks	r - 1 = 3	2,722.92	907.6388 (1)	(1)/(3)
Varieties	p - 1 = 2	147,042.54	73521.27 (2)	(2)/(3)
Error (I)	(r - 1)(p - 1) = 6	18,634.96	3105.83 (3)	
Manures	q - 1 = 3	170,990.25	56996.75 (4)	(4)/(6)
Variety x Manure	(p - 1)(q - 1) = 6	64,581.63	10763.6 (5)	(5)/(6)
Error (II)	27	25,243.62	934.95 (6)	
Total	N - 1 = 47	429,215.92		

Since F for interaction 10763.6/934.95 = 11.512 > Fcritical (appx 3.56 at LOS 1%), hypothesis of no interaction effects is rejected. As such, we do not perform the test for main effects of the Factors and hence corresponding F’s are not shown in the table.

$CF = G^2/N = 9970^2/pqr = 9970^2/3x4x4 = 2,070,852.08333$

$\text{Block SS} = (2596^2 + 2392^2 + 2414^2 + 2568^2)/pq – CF$

$= 24882900/12 - 2,070,852.0833 = 2,073,575 - 2,070,852.0833$

$= 2,722.91667$

$\text{Variety SS} = (2092^2 + 3741^2 + 4137^2)/rq – CF$

$= 35486314/16 - 2,070,852.0833 = 2,217,894.625 - 2,070,852.0833$

$= 147,042.5417$

$\text{Manure SS} = (1395^2 + 2565^2 + 2607^2 + 3403^2)/rp – 9970^2/48$

$= 26902108/12 – 2,070,852.08333 = 170,990.25$

$\text{Variety x Manure SS} = (324^2 + 665^2 +…. 1009^2 + 1616^2)/r – 9970^2/48$

$– \text{Manure SS} – \text{Variety SS}$

$= 9813866/4 – 2,388,884.875 =64,581.625$

$\text{Raw Total SS} = \Sigma y^2 =$ Sum of squares of 48 values from Variety-Manure square

$= 94^2 + 440^2 +…. + 323^2 + 450^2 = 2,500,068$

$\text{Total SS} = RSS – CF = 2,500,068 - 2,070,852.0833 = 429,215.91667$

$\text{Error (II) SS} = \text{Total SS} – \text{Variety SS} – \text{Var x Man SS} – \text{Man SS} – \text{Block SS}$

$= 25,243.62493$ (By subtraction)

Advantages and Disadvantages

The split-plot design has two errors, of which E_{II} is smaller than E_I. Hence usually, the B and AB effects will be estimated and tested more precisely than the A effects. The main advantage of the design is that often it is possible to introduce the second factor B, requiring small experimental material, along with A in a split-plot arrangement at little extra cost. If we have a choice for the allocation of factor A and factor B to the whole plots and split-plots, we shall apply the factor which is more important to the split-plot.

The disadvantages of this design are that the presence of two errors makes the analysis difficult and sometimes the error E_I may be too large.

Although the experimental error for sub-plot treatments and interaction is smaller than that for whole-plot treatment, it can be shown that the average experimental error over all treatment comparisons is the same for a split-plot design and the corresponding factorial experiment in an RBD.

In summary, when one of the treatment factors needs more replication or experimental units (material) than another or when it is hard to change the level of one of the factors, these design become important. The primary disadvantage of these designs is the loss in precision in the whole plot treatment comparison and the statistical complexity.

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

BSDCHZC355
Statistical Inference & Applications

Shaibal Kr. Sen
Session 13, 14 &15

innovate achieve lead

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

STATISTICAL INFERENCES

&

APPLICATIONS

Scheduled Topics to be covered during Session 13, 14 & 15

Statistical Quality Control -

- Importance of SQC in industry.
- Statistical basis of Shewhart control charts.
- Construction of control charts for variables (mean, range and standard deviation) and attributes (p , np, & c- charts with fixed and varying sample sizes).
- Interpretation of control charts.
- Natural tolerance limits and specification limits,
- Process Capability Index.

Concept of Six Sigma and its importance.

What are the determinants for purchasing a product (or availing a service) by a customer?

Necessary Requirement of the product or service for the customer –

Acceptable Quality of the product or service.

Affordable Price of product or service.

Importance of Quality : Controlling and improving quality has become an important business strategy for many organizations; Manufacturers, Distributors, Transportation Companies, Financial Services organizations; Health care providers and Government agencies. Quality is a competitive advantage. A business that can delight customers by improving and controlling quality can dominate its competitors.

Quality : We may define quality in many ways. Most people have a conceptual understanding of quality as relating to one or more desirable characteristics that a product or service should possess. Although this conceptual understanding is certainly a useful starting point, we will give a more precise and useful definition.

Definition : Quality means fitness for use.

Quality characteristics : Every product possesses a number of elements that jointly describe what the user or consumer thinks about quality. These parameters are often called quality characteristics. Sometimes these are called Critical-to-Quality (CTQ) characteristics. Quality characteristics may be of several types –

1. Physical : Length, Weight, Voltage, Viscosity.
2. Sensory : Taste, Appearance, Colour
3. Time Orientation : Reliability, Durability, Serviceability

Dimensions of Quality **:**

(1) Performance – Will the product do the intended job?

(2) Reliability – How often does the product fail?

(3) Durabilty – How long does the product last?

(4) Serviceability – How easy is it to repair the product?

(5) Aesthetics – What does the product look like?

(6) Features – What does the product do?

(7) Perceived Quality – What is the reputation of the company or its product?

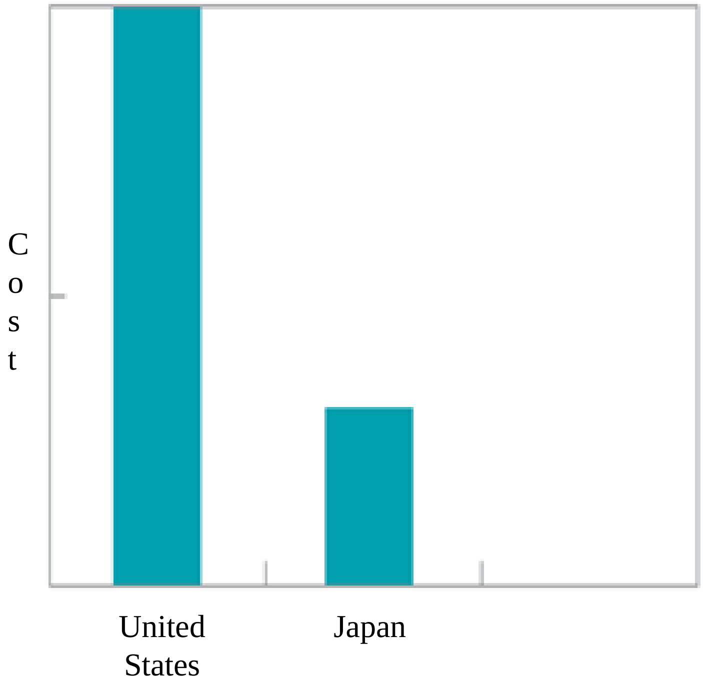
(8) Conformance to Standards – Is the product made exactly as the designer intended?

Definition **:** Quality is inversely proportional to variability. This definition implies that if the variability in the important characteristics of product decreases, the quality of product increases. (Note : Here variability is referred as unwanted or harmful variability).

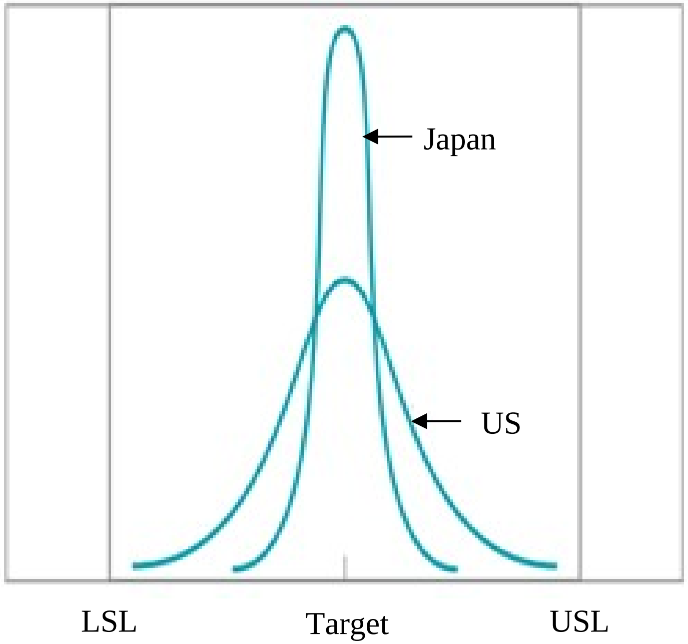
Quality Engineering **:** is the set of Operational, Managerial & Engineering activities that a company uses to ensure that the quality characteristics of the product are at nominal or required levels and that the variability around these desired levels is minimum.

Quality improvement is the reduction of variability in processes and products.

Most organizations find it difficult and expensive to provide the customer with products that have quality characteristics that are always identical from unit to unit or are at levels that match customer expectations. A major reason for this is variability. There is certain amount of variability in every product; consequently, no two products are ever identical. For example, the thickness of the blades on a jet turbine engine impeller is not identical even on the same impeller. Blade thickness will also differ between impellers. If this variation in blade thickness is small, then it may have no impact on the customer. However, if the variation is large, then the customer may perceive the unit to be undesirable and unacceptable. Sources of this variability include differences in materials, differences in the performance and operation of the manufacturing equipment and differences in the way the operators perform their tasks. This line of thinking led to the previous definition of quality improvement.



Warranty Costs for Transmissions



Distributions of Critical Dimensions for Transmiossion

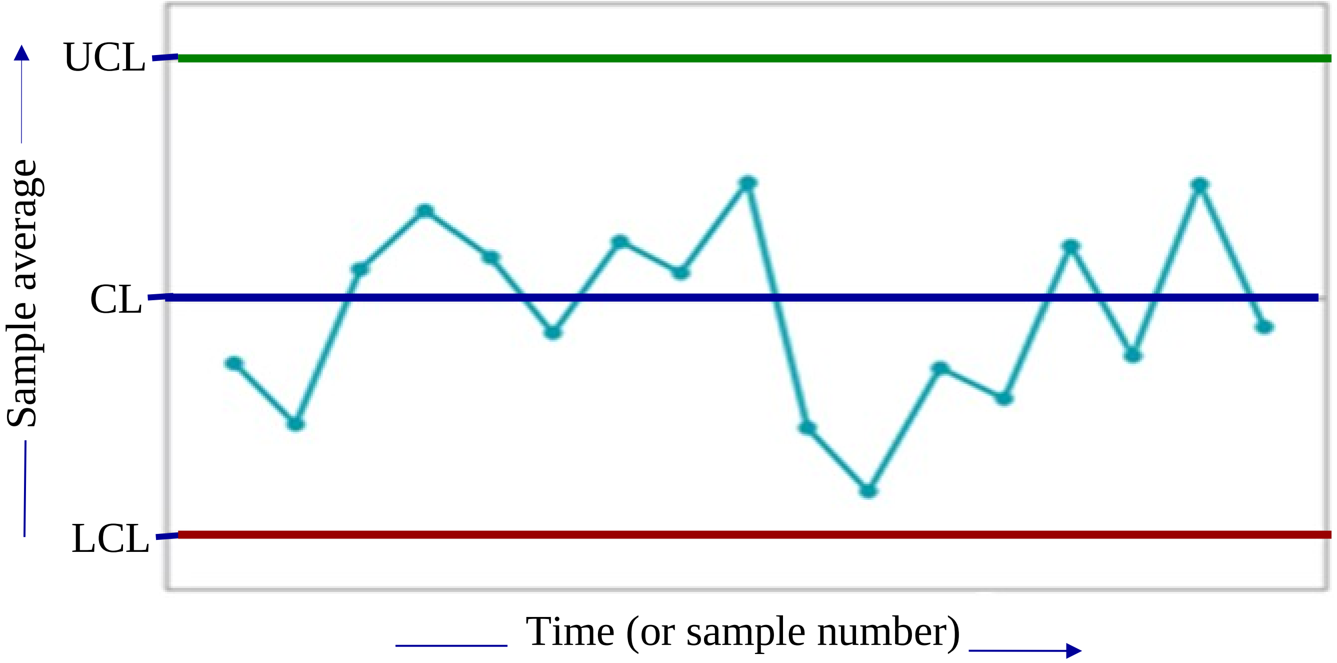
Since Variability can only be described in statistical terms, Statistical Methods play a central role in Quality improvement efforts. In the application of statistical methods to quality engg., it is fairly typical to classify data on quality characteristics as either attributes or variables data. Variables data are usually Continuous Measurements, such as length, voltage or viscosity. Attributes data, on the other hand, are usually Discrete Data, often taking the form of counts. Such as the loan applications that could not be properly processed because of missing required information, or the number of emergency room arrivals that have to wait more than 30 minutes to receive medical attention. We will describe statistical-based quality engg. tools for dealing with both types of data.

Statistical methods and their application in quality improvement have had a long history. In 1924, Walter A. Shewhart of the Bell Telephone Laboratories developed the Statistical Control Chart concept, which is often considered the formal beginning of Statistical Quality Control (SQC). Towards the end of 1920s, Harold F. Dodge and Harry G. Romig, both of Bell Telephone Laboratories, developed statistically based acceptance sampling as an alternative to 100% inspection. By the middle of the 1930s, statistical quality control methods were in wide use at Western Electric, the manufacturing arm of the Bell System.

A value of measurement that corresponds to the desired value for the quality characteristic is called the ‘Nominal’ or ‘Target Value’ for that characteristic. These target values are usually bounded by a range of values that, most typically, we believe will be sufficiently close to the target so as to not impact the function or performance of the product if the quality characteristic is in that range. The largest allowable value for a quality characteristic is called the Upper Specification Limit (USL) and the smallest allowable value for a quality characteristic is called the Lower Specification Limit (LSL). Some quality characteristics have specification limits on only one side of the target. For example, the compressive strength of a component used in an automobile bumper likely has a target value and a lower specification limit, but not an upper specification limit.

A Control Chart is one of the primary techniques of Statistical Process Control (SPC). A typical control chart is shown here. This chart plots the averages of measurements of quality characteristic in samples taken from process versus time (or the sample number). The chart has a central line (CL) and upper & lower control limits (UCL & LCL) as shown in the figure. The central line shows where this process characteristic should fall if there are no unusual sources of variability present. The control limits are determined from some simple charts are applied to the output variable (s) in a system as depicted in the figure. However, in some cases they can be usefully applied to the inputs as well.

The control chart is a very useful process monitoring technique; when unusual sources of variability are present, sample averages will plot outside the control limits. This is a signal that some investigation of the process should be made and corrective action to remove these unusual sources of variability taken. Systematic use of a control chart is an excellent way to reduce variability.



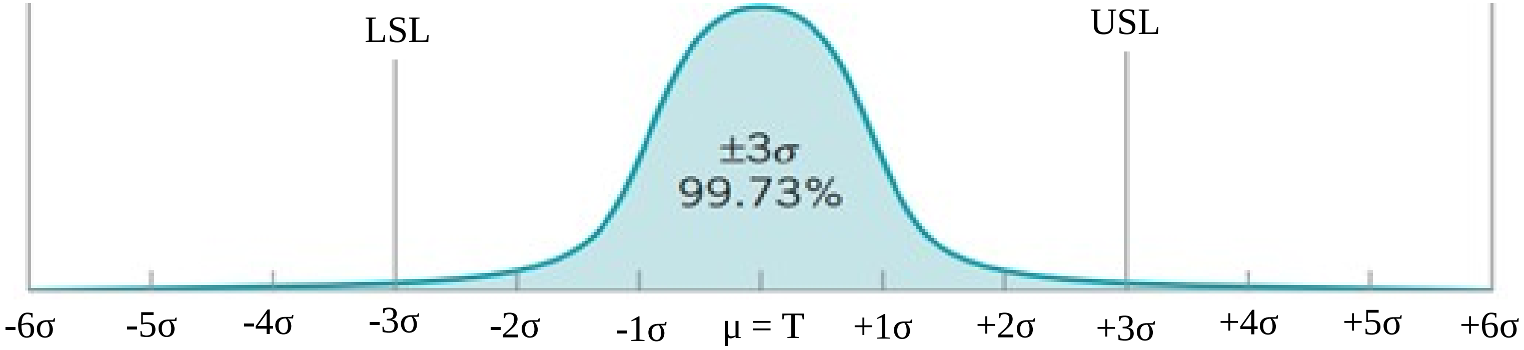
A typical Control Chart

Management Aspects of Quality Improvement

Statistical techniques, including Statistical Process Control (SPC) and Designed Experiments, along with other problem-solving tools are the technical basis for Quality Control and Improvement. However, to be used most effectively, these techniques must be implemented within and be part of a management system that is focused on quality improvement. The management system of an organization must be organized to properly direct the overall quality improvement philosophy and ensure its deployment in all aspects of the business. The effective management of quality involves successful execution of three activities : Quality Planning, Quality Assurance and Quality Control & Improvement.

Six-Sigma

Products with many components typically have many possibilities for failure or defects to occur. Motorola developed the Six-Sigma Programme in the year 1980 as a response to the demand for their products. The focus of Six-Sigma is reducing variability in key product quality characteristics to the level at which failure or defects are extremely unlikely.


Normal Distribution centered at target T

Spec. Limit	% inside Spec	% Defective	PPM Defective
±1σ	68.27	31.73	317300
±2σ	95.45	4.55	45500
±3σ	99.73	0.27	2700
±4σ	99.9937	0.0063	63
±5σ	99.999943	0.000057	0.57
±6σ	99.9999998	0.0000002	0.002

Figure in slide shows a normal probability distribution as a model for a quality characteristic with the specification limits at three standard deviations on either side of the mean. Now it turns out that in this situation the probability of producing a product within the specification is 0.9973 (or 99.73%), which corresponds to 2700 parts per million (PPM) defective. This is referred to as three-sigma quality performance, and it actually sounds pretty good. However, suppose we have a product that consists of an assembly of 100 independent components or parts and all 100 of these parts must be non-defective for the product to function satisfactorily. The probability that any specific unit of product is non-defective is -

$0.9973 \times 0.9973 \times \ldots \times 0.997 = (0.9973)^{100} = 0.7631$

That is about 23.69% (ie. 100 – 76.31) of the products produced under this is not acceptable situation, because many products used by to-day’s society are made up of many components. Even a relatively simple service activity, such as a visit by a family of four to a fast-food restaurant, can involve the assembly of several dozen components. A typical automobile has about 100,000 components and an airplane has between one and two million!

The Motorola six-sigma concept is to reduce the variability in the process so that the specification limits are at least six standard deviation from the mean. Then, as shown in the figure, there will only be about 2 parts per billion defective. Under Six-Sigma Quality, the probability that any specific unit of the hypothetical product above is non-defective is 0.9999998 or 0.002 PPM defective, a much better situation.

Shewhart's Control Charts

Chance & Assignable causes of Quality Variation **:** In any production process, regardless of how well designed or carefully maintained it is, a certain amount of inherent or natural variability will always exist. This natural variability or "background noise" is the cumulative effect of many small, essentially unavoidable causes. In the framework of SQC, this natural variability is often called a "stable system of chance causes". A process that is operating with only Chance Causes of Variation present is said to be in Statistical Control. In other words, the chance causes are inherent part of of the process.

Other kinds of variability may occasionally be present in the output of process. This variability in key quality characteristics usually arises from three sources – improperly adjusted or uncontrolled machines, operator errors or defective raw material. Such variability is generally large when compared to the background noise and it usually represents

Shewhart's Control Charts

an unacceptable level of process performance. We refer to these sources of variability that are not part of the chance cause pattern as Assignable Causes of Variation. A process that is operating in the presence of assignable causes is said to be an Out-of-Control process.

Popularly there are seven SPC tools – (i) Histogram, (ii) Check sheet, (iii)Pareto chart, (iv) Cause-and-effect diagram,
(v) Stratification (alternately, flow chart), (vi) Scatter diagram and
(vii) Control charts.

Shewhart control chart is probably the most technically sophisticated. It was developed in the 1920s by Walter A. Shewhart of the Bell Telephone Laboratories. To understand the statistical concepts that form the basis of SPC, we must first describe Shewhart's theory of variability – (i) Chance and (ii) Assignable Causes.

Chance and Assignable Causes of Quality Variation : In any production process, regardless of how well designed or carefully maintained it is, a certain amount of inherent or natural variability will always exist. This natural variability or “background noise” will always exist. This natural variability or “background noise” is the cumulative effect of many small, essentially unavoidable causes. In the framework of statistical quality control, this natural variability is often called a “stable system of chance causes.” A process that is operating with only chance causes of variation present is said to be in statistical control. In other words, the chance causes are an inherent part of the process.

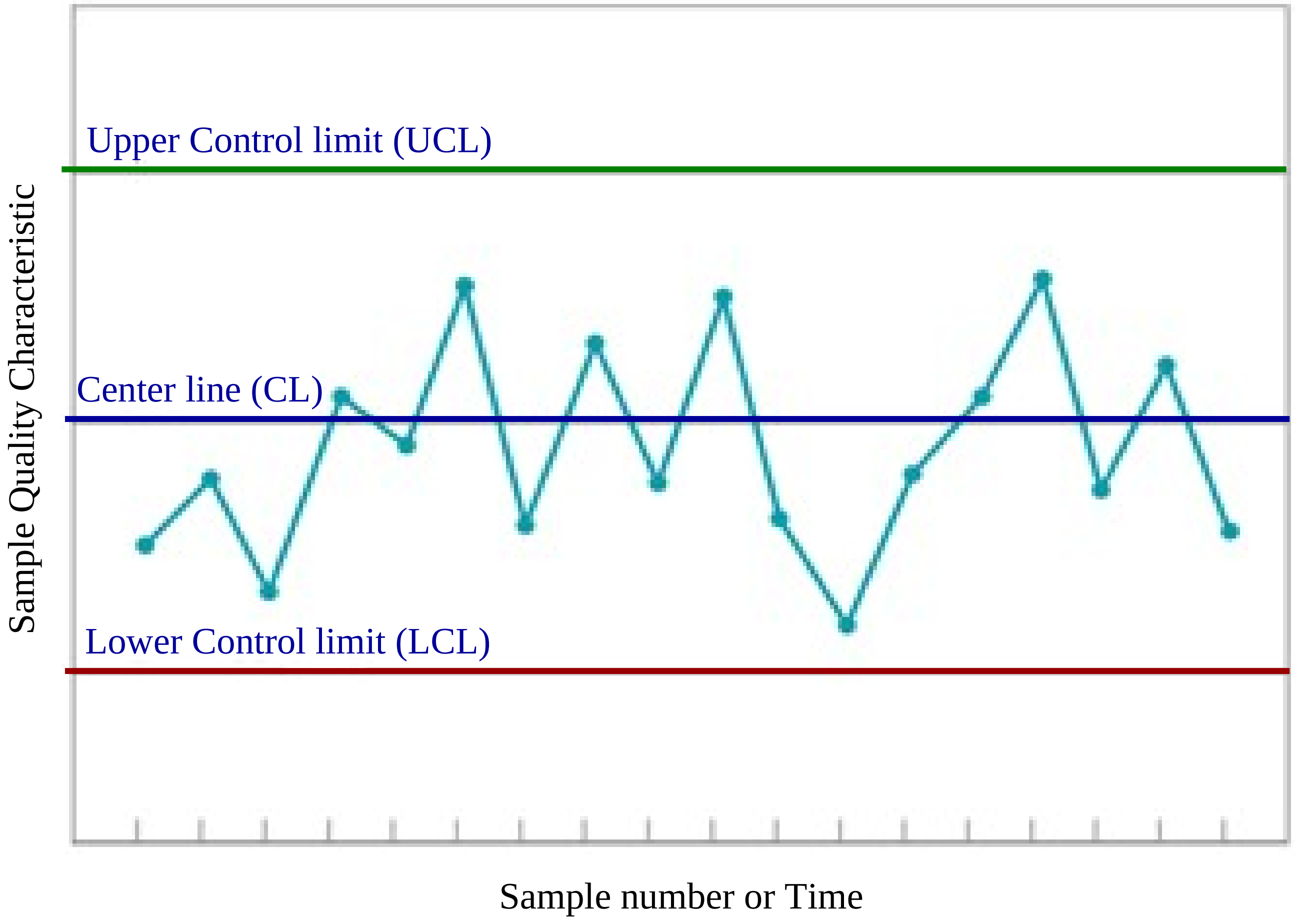
Other kinds of variability, Assignable causes of variation, may occasionally be present in the output of a process. This variability in key quality characteristics usually arises from three sources :
(i) Improperly adjusted or uncontrolled machines, (ii) operator errors,

or (iii) defective raw material. Such variability is generally large when compared to the background noise and it usually represents an unacceptable level of process performance. A process that is operating in the presence of assignable causes is said to be an out-of-control process.

Statistical Basis of the Control Chart

A typical control chart is shown here. The control chart is a graphical display of quality characteristic that has been measured or computed from a sample versus the sample number or time. The chart contains a Center Line (CL) that represents the average value of the quality characteristic corresponding to the in-control state (that is, only chance causes are present). Two other horizontal lines – called the Upper Control Limit (UCL) and the Lower Control Limit (LCL), are also shown on the chart. These control limits are chosen so that if the process is in control, nearly all the sample points will fall between them. As long as the points plot within the control limits, the process is assumed to be in control and no action is necessary. However, a point that plots outside of the control limits is interpreted as evidence that the process is out of control and investigation and corrective action are required to find and eliminate the assignable cause or causes responsible for this behaviour. It is customary to connect the sample points on the control chart.

A typical Control Chart



We may give a general model for a control chart. Let w be a sample statistic that measures quality characteristic of interest and suppose that the mean of w is μ_w and the standard deviation is σ_w. Then the center line, Upper Control Limit (UCL) and Lower Control Limit (LCL) become –

$UCL = \mu_w + L\sigma_w = \mu_w + 3\sigma_w$

Center line = μ_w

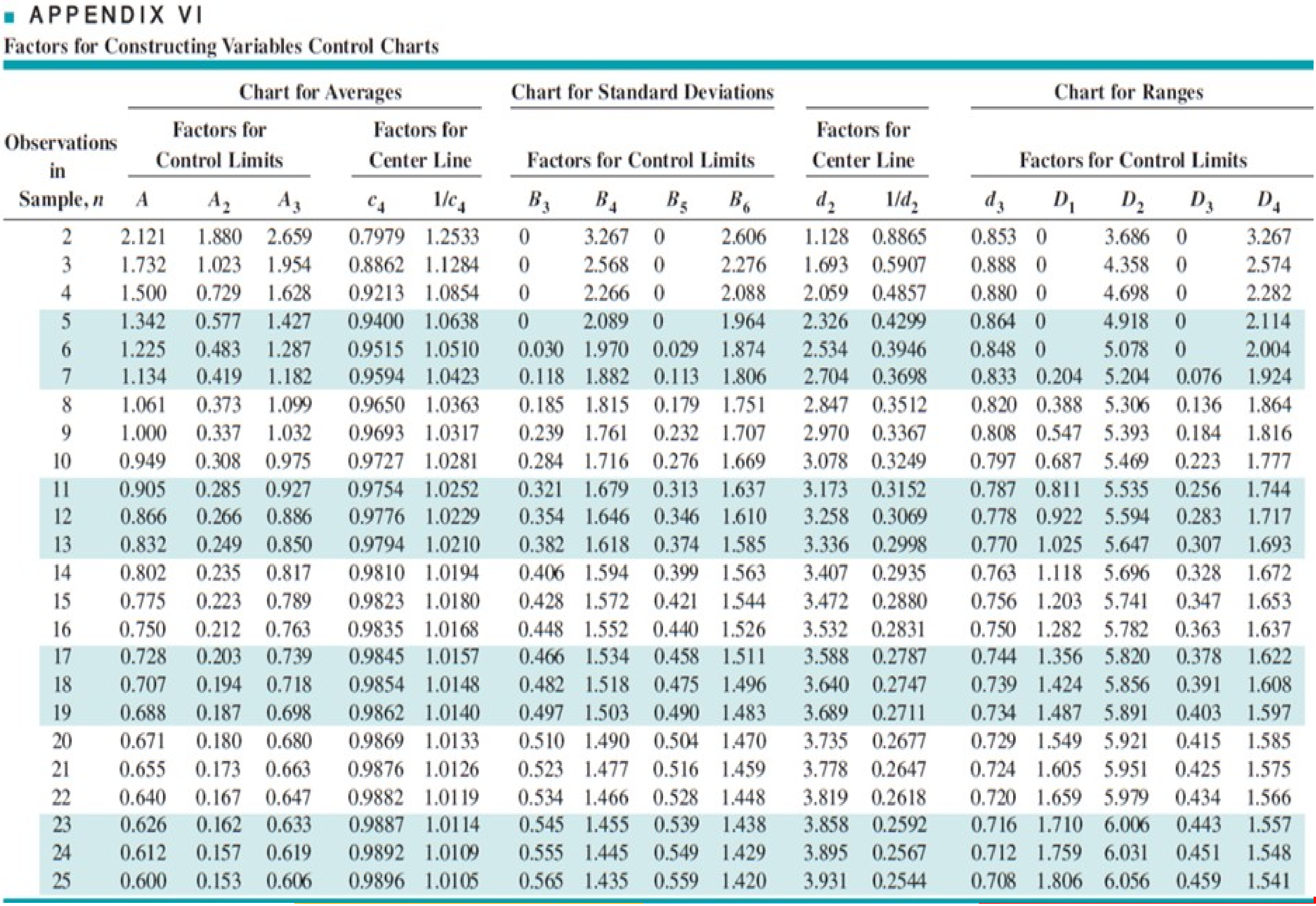
$LCL = \mu_w – L\sigma_w = \mu_w - 3\sigma_w$

where L is the distance of the control limits from the center line, expressed in standard deviation units. This general theory of control charts was first proposed by Walter A. Shewhart and control charts developed according to these principles are often called Shewhart control charts.

$$UCL/LCL = \mu \pm 3\sigma$$
$$UWL/LWL = \mu \pm 2\sigma$$

UCL - Upper Control Limit
LCL - Lower Control Limit
UWL - Upper Warning Limit
LWL - Lower Warning Limit

This table will be referred to solve the numericals



APPENDIX VI

Factors for Constructing Variables Control Charts

Observations in Sample, n	Chart for Averages					Chart for Standard Deviations						Chart for Ranges				
	Factors for Control Limits			Factors for Center Line		Factors for Control Limits				Factors for Center Line		Factors for Control Limits				
	A	A_2	A_3	c_4	$1/c_4$	B_3	B_4	B_5	B_6	d_2	$1/d_2$	d_3	D_1	D_2	D_3	D_4
2	2.121	1.880	2.659	0.7979	1.2533	0	3.267	0	2.606	1.128	0.8865	0.853	0	3.686	0	3.267
3	1.732	1.023	1.954	0.8862	1.1284	0	2.568	0	2.276	1.693	0.5907	0.888	0	4.358	0	2.574
4	1.500	0.729	1.628	0.9213	1.0854	0	2.266	0	2.088	2.059	0.4857	0.880	0	4.698	0	2.282
5	1.342	0.577	1.427	0.9400	1.0638	0	2.089	0	1.964	2.326	0.4299	0.864	0	4.918	0	2.114
6	1.225	0.483	1.287	0.9515	1.0510	0.030	1.970	0.029	1.874	2.534	0.3946	0.848	0	5.078	0	2.004
7	1.134	0.419	1.182	0.9594	1.0423	0.118	1.882	0.113	1.806	2.704	0.3698	0.833	0.204	5.204	0.076	1.924
8	1.061	0.373	1.099	0.9650	1.0363	0.185	1.815	0.179	1.751	2.847	0.3512	0.820	0.388	5.306	0.136	1.864
9	1.000	0.337	1.032	0.9693	1.0317	0.239	1.761	0.232	1.707	2.970	0.3367	0.808	0.547	5.393	0.184	1.816
10	0.949	0.308	0.975	0.9727	1.0281	0.284	1.716	0.276	1.669	3.078	0.3249	0.797	0.687	5.469	0.223	1.777
11	0.905	0.285	0.927	0.9754	1.0252	0.321	1.679	0.313	1.637	3.173	0.3152	0.787	0.811	5.535	0.256	1.744
12	0.866	0.266	0.886	0.9776	1.0229	0.354	1.646	0.346	1.610	3.258	0.3069	0.778	0.922	5.594	0.283	1.717
13	0.832	0.249	0.850	0.9794	1.0210	0.382	1.618	0.374	1.585	3.336	0.2998	0.770	1.025	5.647	0.307	1.693
14	0.802	0.235	0.817	0.9810	1.0194	0.406	1.594	0.399	1.563	3.407	0.2935	0.763	1.118	5.696	0.328	1.672
15	0.775	0.223	0.789	0.9823	1.0180	0.428	1.572	0.421	1.544	3.472	0.2880	0.756	1.203	5.741	0.347	1.653
16	0.750	0.212	0.763	0.9835	1.0168	0.448	1.552	0.440	1.526	3.532	0.2831	0.750	1.282	5.782	0.363	1.637
17	0.728	0.203	0.739	0.9845	1.0157	0.466	1.534	0.458	1.511	3.588	0.2787	0.744	1.356	5.820	0.378	1.622
18	0.707	0.194	0.718	0.9854	1.0148	0.482	1.518	0.475	1.496	3.640	0.2747	0.739	1.424	5.856	0.391	1.608
19	0.688	0.187	0.698	0.9862	1.0140	0.497	1.503	0.490	1.483	3.689	0.2711	0.734	1.487	5.891	0.403	1.597
20	0.671	0.180	0.680	0.9869	1.0133	0.510	1.490	0.504	1.470	3.735	0.2677	0.729	1.549	5.921	0.415	1.585
21	0.655	0.173	0.663	0.9876	1.0126	0.523	1.477	0.516	1.459	3.778	0.2647	0.724	1.605	5.951	0.425	1.575
22	0.640	0.167	0.647	0.9882	1.0119	0.534	1.466	0.528	1.448	3.819	0.2618	0.720	1.659	5.979	0.434	1.566
23	0.626	0.162	0.633	0.9887	1.0114	0.545	1.455	0.539	1.438	3.858	0.2592	0.716	1.710	6.006	0.443	1.557
24	0.612	0.157	0.619	0.9892	1.0109	0.555	1.445	0.549	1.429	3.895	0.2567	0.712	1.759	6.031	0.451	1.548
25	0.600	0.153	0.606	0.9896	1.0105	0.565	1.435	0.559	1.420	3.931	0.2544	0.708	1.806	6.056	0.459	1.541

Types of Control Charts –

I. Charts for Variables – a) Average chart (x-bar chart)
b) Range chart (R chart)

II. Chart for Attributes – a) Charts for defective items –
i) Fraction defective (Varied sample size) p chart
ii) No. of defective (Constant sample size) np chart

b) Charts for defects per unit –
i) Varied sample size u chart
ii) Constant sample size c chart

There are two types of control charts that we deal with.

Variables Control Charts : These charts are applied to data that follow a continuous distribution. Variables include length, thickness, diameter, breaking strength, temperature, viscosity etc. It is used to monitor characteristics that can be measured and have a continuum of values, such as height, weight, or volume.

Attributes Control Charts : These charts are applied to data that follow a discrete distribution. Attributes deal with qualitative information, eg. 40 ± 0.6. Another example of attributes data is the count of defects. A control chart for attributes is used to monitor characteristics that have discrete values and can be counted.

Classifications such as conforming and nonconforming are commonly used in quality control.

Control Chart of Variables

Mean Chart

Range Chart

Standard deviation Chart

Control Charts for $\bar{x}$ and R

Statistical Basis of the Charts **:** Suppose that a quality characteristic is normally distributed with mean μ and standard deviation σ, where both μ and σ are known. If $x_1, x_2, \ldots, x_n$ is a sample of size n, then the average of this sample is

$\bar{x} = (x_1 + x_2 + \ldots, x_n)/n = \sum x_i/n$ (limit of i is from 1 to n)

and we know that $\bar{x}$ is normally distributed with mean μ and standard deviation $\sigma_{\bar{x}} = \sigma/\sqrt{n}$. Furthermore, the probability is 1 – α that any mean will fall between

$$\mu + Z_{\alpha/2}\sigma_{\bar{x}} = \mu + Z_{\alpha/2}(\sigma/\sqrt{n}) \quad \text{and} \quad \mu - Z_{\alpha/2}\sigma_{\bar{x}} = \mu - Z_{\alpha/2}(\sigma/\sqrt{n}) \quad \ldots\ldots \text{(i)}$$

Therefore, if μ and σ are known, equation (i) could be used as Upper & Lower Control Limits on a control chart for a sample means. It is customary to replace $Z_{\alpha/2}$ by 3, so that three sigma limits are employed. If a sample mean falls outside of these limits, it is an indication that the process mean is no longer equal to μ.

In practice, we usually will not know μ and σ. Therefore, they must be estimated from preliminary samples or subgroups taken when the process is thought to be in control. These estimates should usually be based on at least 20 to 25 samples. Suppose that m samples are available, each containing n observations on the quality characteristic. Typically n will be small, often either 4, 5 or 6. These small sample sizes usually result from the construction of rational subgroups and from the fact that the sampling and inspection costs associated with variables measurements are usually relatively large. Let $\bar{x}_1$, $\bar{x}_2$, …., $\bar{x}_m$ be the average of each sample. Then the best estimator of μ, the process average, is the grand average - say,

$$\bar{\bar{x}} = \frac{\bar{x}_1 + \bar{x}_2 + \cdots + \bar{x}_m}{m}$$

Thus, $\bar{\bar{x}}$ would be used as the center line on the $\bar{x}$ chart.

To construct the control limits, we need an estimate of the standard deviation σ.

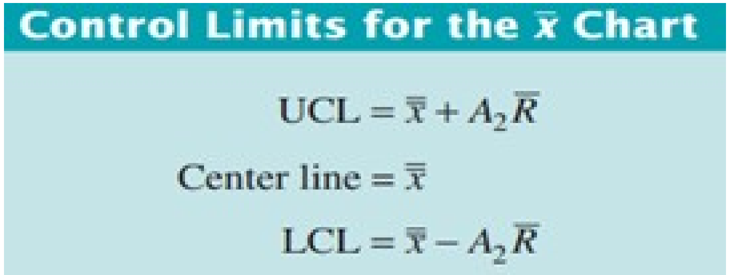
We may estimate σ from either the standard deviations or the ranges of m samples. For the present, we will use the range method. If $x_1, x_2, \ldots, x_n$ is a sample of size n, then the range of the sample is the difference between the largest and smallest observation; that is,

$$R = x_{max} - x_{min}$$

Let $R_1, R_2, \ldots, R_m$ be the ranges of m samples. The average range is

$$\bar{R} = (R_1 + R_2 \ldots. + R_m)/m$$

We may now give the formulas for constructing the control limits on the $\bar{x}$ chart. They are as follows –



Control Limits for the $\bar{x}$ Chart

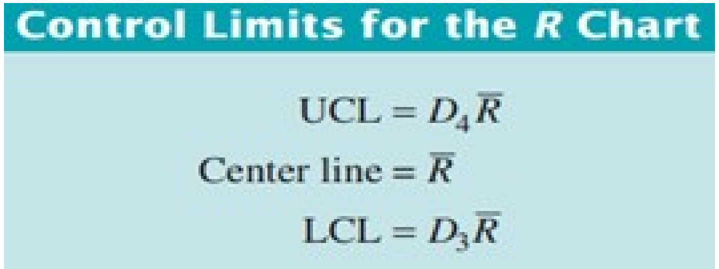
$$\text{UCL} = \bar{\bar{x}} + A_2\bar{R}$$

$$\text{Center line} = \bar{\bar{x}}$$

$$\text{LCL} = \bar{\bar{x}} - A_2\bar{R}$$

The constant A_2 is tabulated for various sample sizes in Appendix Table VI.

Process variability may be monitored by plotting values of the sample range R on a control chart. The center line and control limits of the R chart are as follows -



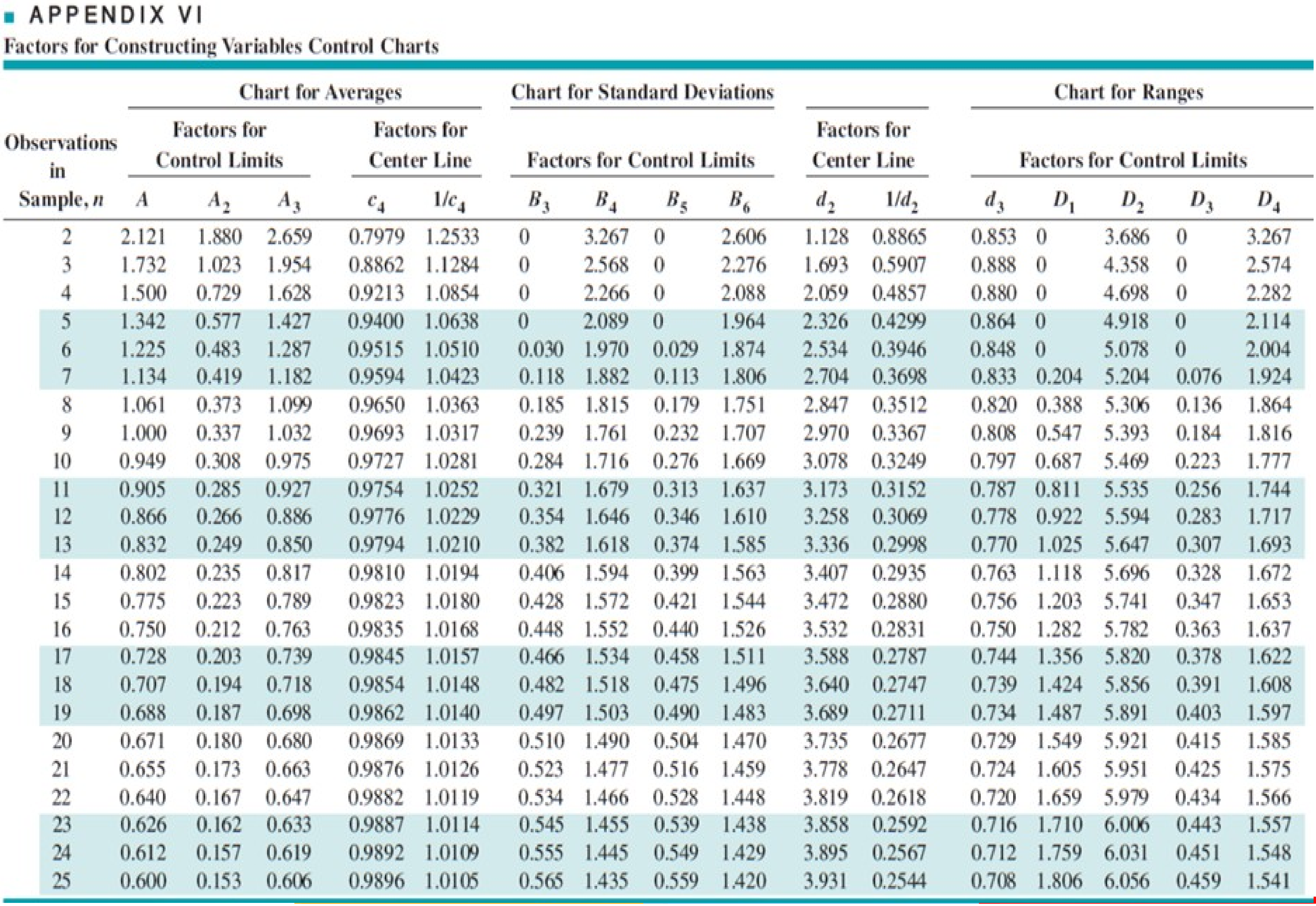
Control Limits for the R Chart

$$\text{UCL} = D_4\bar{R}$$

$$\text{Center line} = \bar{R}$$

$$\text{LCL} = D_3\bar{R}$$

The constant D_3 & D_4 are tabulated for various values of n in Appendix Table VI.



APPENDIX VI

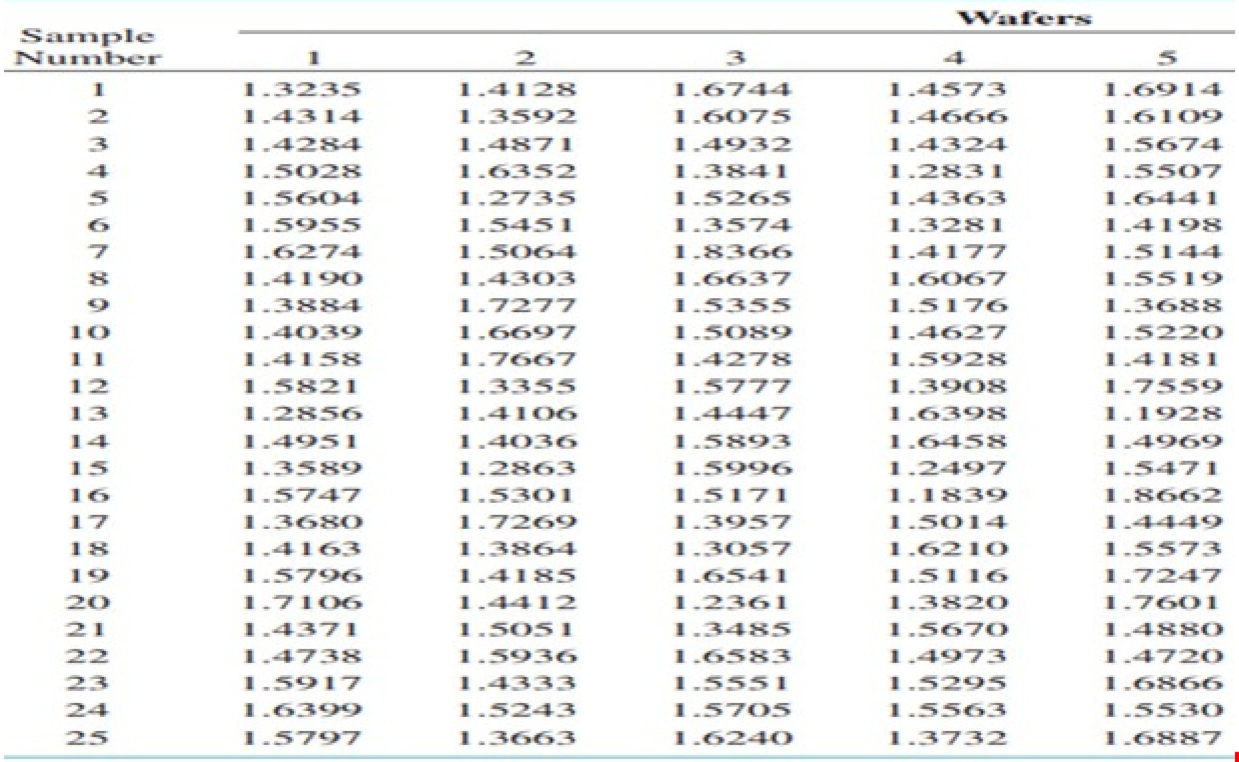
Factors for Constructing Variables Control Charts

Observations in Sample, n	Chart for Averages					Chart for Standard Deviations						Chart for Ranges				
	Factors for Control Limits			Factors for Center Line		Factors for Control Limits				Factors for Center Line		Factors for Control Limits				
	A	A_2	A_3	c_4	$1/c_4$	B_3	B_4	B_5	B_6	d_2	$1/d_2$	d_3	D_1	D_2	D_3	D_4
2	2.121	1.880	2.659	0.7979	1.2533	0	3.267	0	2.606	1.128	0.8865	0.853	0	3.686	0	3.267
3	1.732	1.023	1.954	0.8862	1.1284	0	2.568	0	2.276	1.693	0.5907	0.888	0	4.358	0	2.574
4	1.500	0.729	1.628	0.9213	1.0854	0	2.266	0	2.088	2.059	0.4857	0.880	0	4.698	0	2.282
5	1.342	0.577	1.427	0.9400	1.0638	0	2.089	0	1.964	2.326	0.4299	0.864	0	4.918	0	2.114
6	1.225	0.483	1.287	0.9515	1.0510	0.030	1.970	0.029	1.874	2.534	0.3946	0.848	0	5.078	0	2.004
7	1.134	0.419	1.182	0.9594	1.0423	0.118	1.882	0.113	1.806	2.704	0.3698	0.833	0.204	5.204	0.076	1.924
8	1.061	0.373	1.099	0.9650	1.0363	0.185	1.815	0.179	1.751	2.847	0.3512	0.820	0.388	5.306	0.136	1.864
9	1.000	0.337	1.032	0.9693	1.0317	0.239	1.761	0.232	1.707	2.970	0.3367	0.808	0.547	5.393	0.184	1.816
10	0.949	0.308	0.975	0.9727	1.0281	0.284	1.716	0.276	1.669	3.078	0.3249	0.797	0.687	5.469	0.223	1.777
11	0.905	0.285	0.927	0.9754	1.0252	0.321	1.679	0.313	1.637	3.173	0.3152	0.787	0.811	5.535	0.256	1.744
12	0.866	0.266	0.886	0.9776	1.0229	0.354	1.646	0.346	1.610	3.258	0.3069	0.778	0.922	5.594	0.283	1.717
13	0.832	0.249	0.850	0.9794	1.0210	0.382	1.618	0.374	1.585	3.336	0.2998	0.770	1.025	5.647	0.307	1.693
14	0.802	0.235	0.817	0.9810	1.0194	0.406	1.594	0.399	1.563	3.407	0.2935	0.763	1.118	5.696	0.328	1.672
15	0.775	0.223	0.789	0.9823	1.0180	0.428	1.572	0.421	1.544	3.472	0.2880	0.756	1.203	5.741	0.347	1.653
16	0.750	0.212	0.763	0.9835	1.0168	0.448	1.552	0.440	1.526	3.532	0.2831	0.750	1.282	5.782	0.363	1.637
17	0.728	0.203	0.739	0.9845	1.0157	0.466	1.534	0.458	1.511	3.588	0.2787	0.744	1.356	5.820	0.378	1.622
18	0.707	0.194	0.718	0.9854	1.0148	0.482	1.518	0.475	1.496	3.640	0.2747	0.739	1.424	5.856	0.391	1.608
19	0.688	0.187	0.698	0.9862	1.0140	0.497	1.503	0.490	1.483	3.689	0.2711	0.734	1.487	5.891	0.403	1.597
20	0.671	0.180	0.680	0.9869	1.0133	0.510	1.490	0.504	1.470	3.735	0.2677	0.729	1.549	5.921	0.415	1.585
21	0.655	0.173	0.663	0.9876	1.0126	0.523	1.477	0.516	1.459	3.778	0.2647	0.724	1.605	5.951	0.425	1.575
22	0.640	0.167	0.647	0.9882	1.0119	0.534	1.466	0.528	1.448	3.819	0.2618	0.720	1.659	5.979	0.434	1.566
23	0.626	0.162	0.633	0.9887	1.0114	0.545	1.455	0.539	1.438	3.858	0.2592	0.716	1.710	6.006	0.443	1.557
24	0.612	0.157	0.619	0.9892	1.0109	0.555	1.445	0.549	1.429	3.895	0.2567	0.712	1.759	6.031	0.451	1.548
25	0.600	0.153	0.606	0.9896	1.0105	0.565	1.435	0.559	1.420	3.931	0.2544	0.708	1.806	6.056	0.459	1.541

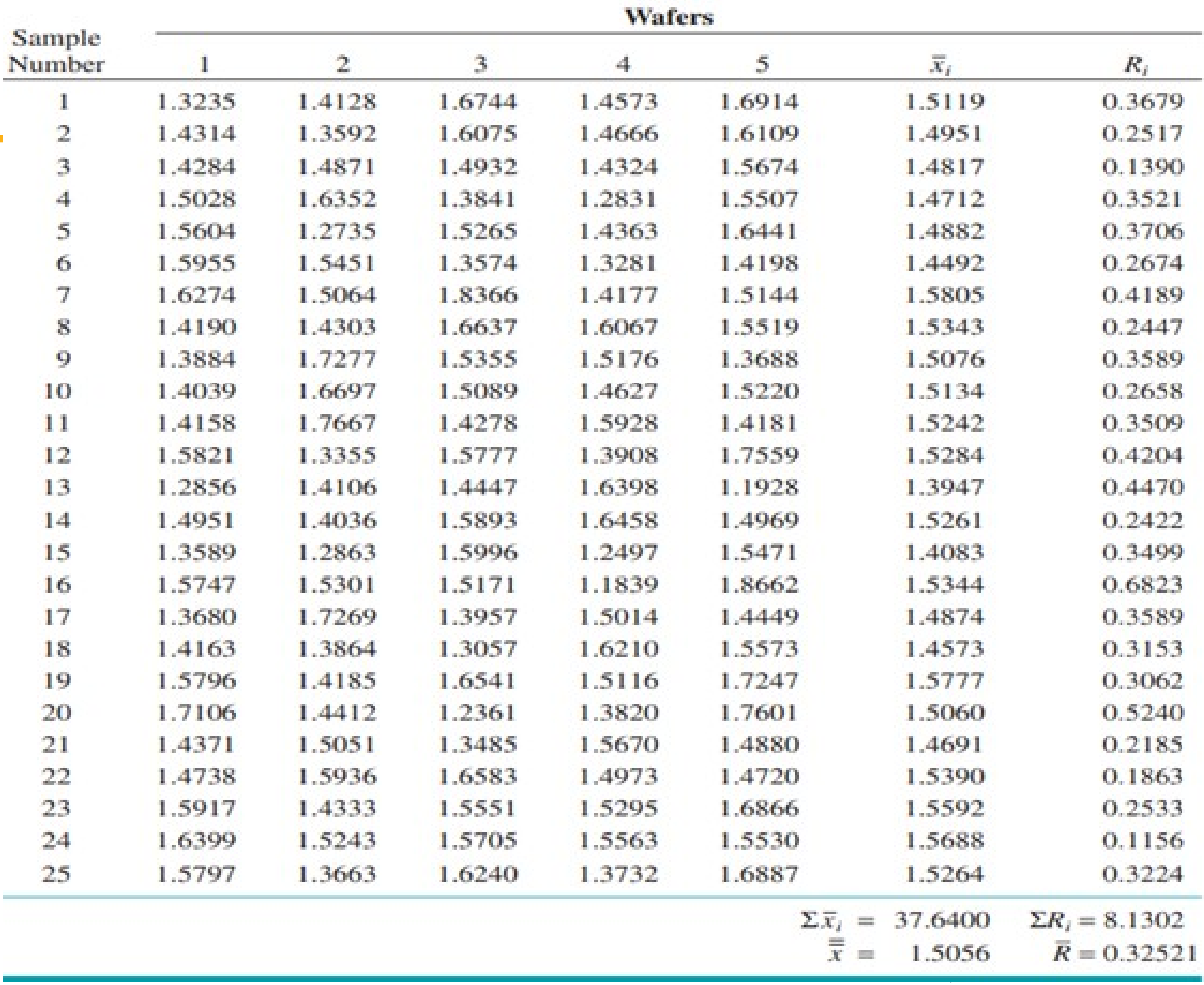
Ex : $\bar{x}$ and R chart for a manufacturing process

A hard-bake process is used in conjunction with photolithography in semiconductor manufacturing. We wish to establish statistical control of the flow width of the resist in this process using $\bar{x}$ and R charts. Twenty five samples, each of size five wafers, have been taken when we think the process is in control. The interval of time between samples or subgroups is one hour. The flow width measurement data (in microns) from these samples is shown in the table.

Flow width measurements (microns) for Hard-Bake Process



Sample Number	Wafers				
	1	2	3	4	5
1	1.3235	1.4128	1.6744	1.4573	1.6914
2	1.4314	1.3592	1.6075	1.4666	1.6109
3	1.4284	1.4871	1.4932	1.4324	1.5674
4	1.5028	1.6352	1.3841	1.2831	1.5507
5	1.5604	1.2735	1.5265	1.4363	1.6441
6	1.5955	1.5451	1.3574	1.3281	1.4198
7	1.6274	1.5064	1.8366	1.4177	1.5144
8	1.4190	1.4303	1.6637	1.6067	1.5519
9	1.3884	1.7277	1.5355	1.5176	1.3688
10	1.4039	1.6697	1.5089	1.4627	1.5220
11	1.4158	1.7667	1.4278	1.5928	1.4181
12	1.5821	1.3355	1.5777	1.3908	1.7559
13	1.2856	1.4106	1.4447	1.6398	1.1928
14	1.4951	1.4036	1.5893	1.6458	1.4969
15	1.3589	1.2863	1.5996	1.2497	1.5471
16	1.5747	1.5301	1.5171	1.1839	1.8662
17	1.3680	1.7269	1.3957	1.5014	1.4449
18	1.4163	1.3864	1.3057	1.6210	1.5573
19	1.5796	1.4185	1.6541	1.5116	1.7247
20	1.7106	1.4412	1.2361	1.3820	1.7601
21	1.4371	1.5051	1.3485	1.5670	1.4880
22	1.4738	1.5936	1.6583	1.4973	1.4720
23	1.5917	1.4333	1.5551	1.5295	1.6866
24	1.6399	1.5243	1.5705	1.5563	1.5530
25	1.5797	1.3663	1.6240	1.3732	1.6887



Sample Number	Wafers 1	2	3	4	5	$\bar{x}_i$	R_i
1	1.3235	1.4128	1.6744	1.4573	1.6914	1.5119	0.3679
2	1.4314	1.3592	1.6075	1.4666	1.6109	1.4951	0.2517
3	1.4284	1.4871	1.4932	1.4324	1.5674	1.4817	0.1390
4	1.5028	1.6352	1.3841	1.2831	1.5507	1.4712	0.3521
5	1.5604	1.2735	1.5265	1.4363	1.6441	1.4882	0.3706
6	1.5955	1.5451	1.3574	1.3281	1.4198	1.4492	0.2674
7	1.6274	1.5064	1.8366	1.4177	1.5144	1.5805	0.4189
8	1.4190	1.4303	1.6637	1.6067	1.5519	1.5343	0.2447
9	1.3884	1.7277	1.5355	1.5176	1.3688	1.5076	0.3589
10	1.4039	1.6697	1.5089	1.4627	1.5220	1.5134	0.2658
11	1.4158	1.7667	1.4278	1.5928	1.4181	1.5242	0.3509
12	1.5821	1.3355	1.5777	1.3908	1.7559	1.5284	0.4204
13	1.2856	1.4106	1.4447	1.6398	1.1928	1.3947	0.4470
14	1.4951	1.4036	1.5893	1.6458	1.4969	1.5261	0.2422
15	1.3589	1.2863	1.5996	1.2497	1.5471	1.4083	0.3499
16	1.5747	1.5301	1.5171	1.1839	1.8662	1.5344	0.6823
17	1.3680	1.7269	1.3957	1.5014	1.4449	1.4874	0.3589
18	1.4163	1.3864	1.3057	1.6210	1.5573	1.4573	0.3153
19	1.5796	1.4185	1.6541	1.5116	1.7247	1.5777	0.3062
20	1.7106	1.4412	1.2361	1.3820	1.7601	1.5060	0.5240
21	1.4371	1.5051	1.3485	1.5670	1.4880	1.4691	0.2185
22	1.4738	1.5936	1.6583	1.4973	1.4720	1.5390	0.1863
23	1.5917	1.4333	1.5551	1.5295	1.6866	1.5592	0.2533
24	1.6399	1.5243	1.5705	1.5563	1.5530	1.5688	0.1156
25	1.5797	1.3663	1.6240	1.3732	1.6887	1.5264	0.3224
						$\Sigma\bar{x}_i = 37.6400$	$\Sigma R_i = 8.1302$
						$\bar{\bar{x}} = 1.5056$	$\bar{R} = 0.32521$

When setting up $\bar{x}$ and R charts, it is best to begin with the R chart because the control limits on the $\bar{x}$ chart depend on the process variability, unless process variability is in control, these limits will not have much meaning. Using the data in the table of the exercise, we find the center line for R chart is

$$\bar{R} = \{ \sum_{i=1}^{25} (R_i)/25\} = 8.1302/25 = 0.32521$$

For samples of n = 5, we find from Appendix Table VI that $D_3 = 0$ and $D_4 = 2.114$. Therefore, the control limits for the R chart are

$$LCL = \bar{R}D_3 = 0.32521(0) = 0$$

$$UCL = \underline{R}D_4 = 0.32521(2.114) = 0.68749$$

The R chart is shown in subsequent slide. Both control charts were constructed by Minitab which uses more decimal places in the calculation than we did. When the 25 sample ranges are plotted on the R chart there is no indication of an out-of-control condition.

Since R chart indicates that process variability is in control, we may now construct $\bar{x}$ chart. The center line is

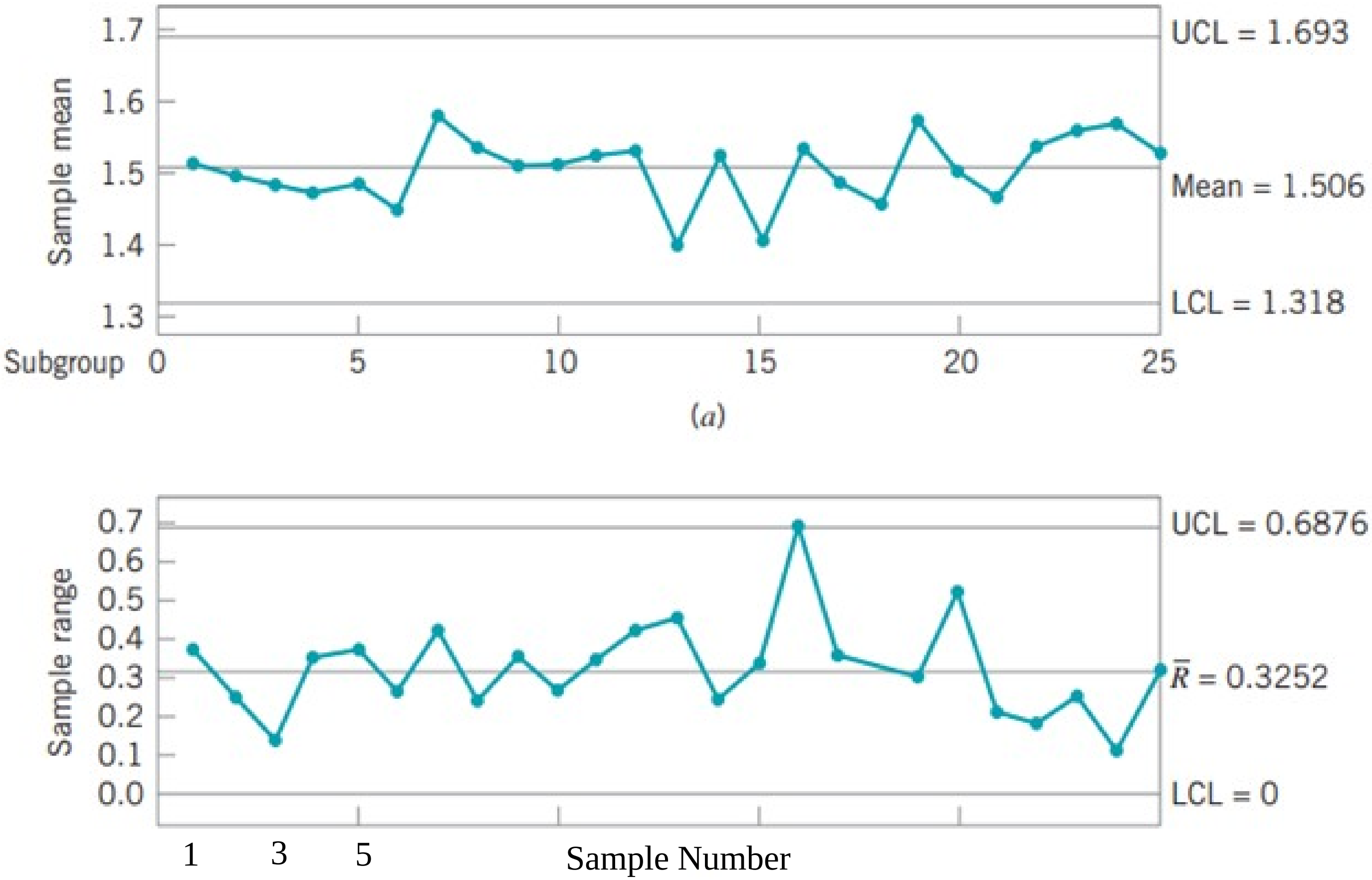
$$\bar{\bar{x}} = (\Sigma \bar{x}_i)/25 = 37.64/25 = 1.5056$$ (Summation limit i = 1 to 25)

To find the control limits on the $\bar{x}$ chart, we use $A_2 = 0.577$ from Appendix Table VI for sample of size n = 5 and the equation of control limit for x chart to find

$$UCL = \bar{\bar{x}} + A_2\bar{R} = 1.5056 + (0.577)(0.32521) = 1.69325$$ and

$$UCL = \bar{\bar{x}} - A_2\bar{R} = 1.5056 - (0.577)(0.32521) = 1.31795$$

The $\bar{x}$ chart is also shown. When the preliminary sample averages are plotted on this chart, no indication of an out-of-control condition is observed. Therefore, since $\bar{x}$ and R charts exhibit control, we would conclude that the process is in control at the stated levels and adopt the trial control limits for use in phase II, where monitoring of future production is of interest.



(b)

$\bar{x}$ (a) and R charts (b) (from Minitab) for flow width in hard-bake process

Interpretation of $\bar{x}$ and R charts

If all points fall in between UCL and LCL in both mean and range charts without showing in specific pattern then the process is said to be in the state of under statistical control otherwise the process is lack or out of statistical control.

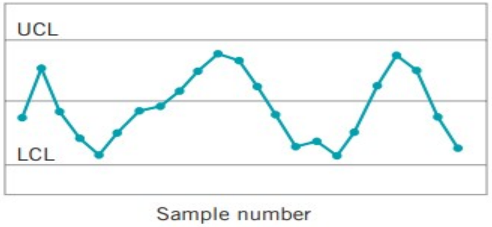
Lack of control in Mean chart indicates machine problems.

Lack of control in Range chart indicates workers problems.

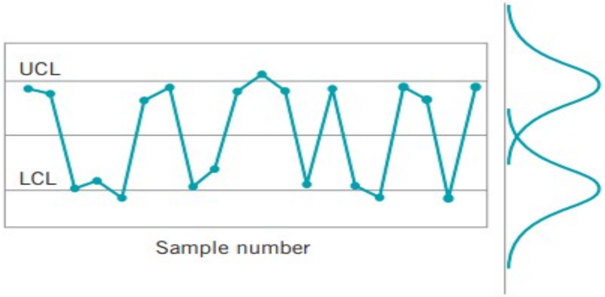
Cases of Lack of control :

If at least one point falls outside UCL and / or LCL.

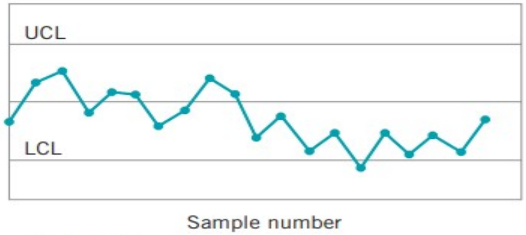
If all points fall in between the limits but showing specific patterns which are shown in the next slide.



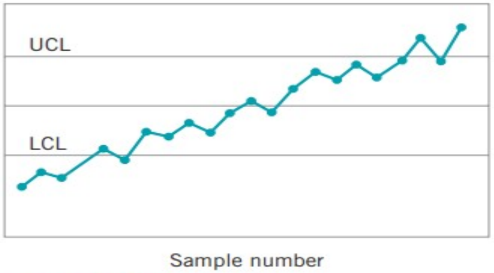
■ FIGURE 6.8 Cycles on a control chart.



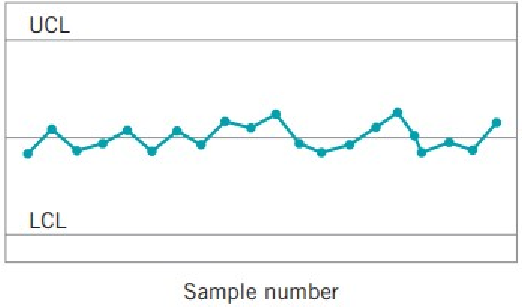
■ FIGURE 6.9 A mixture pattern.



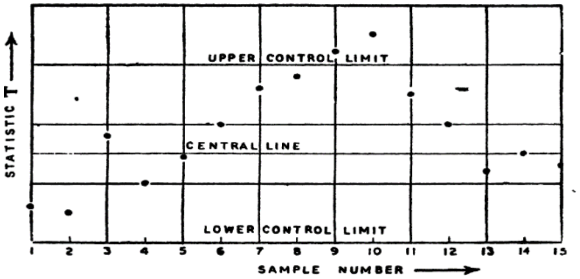
■ FIGURE 6.10 A shift in process level.



■ FIGURE 6.11 A trend in process level.



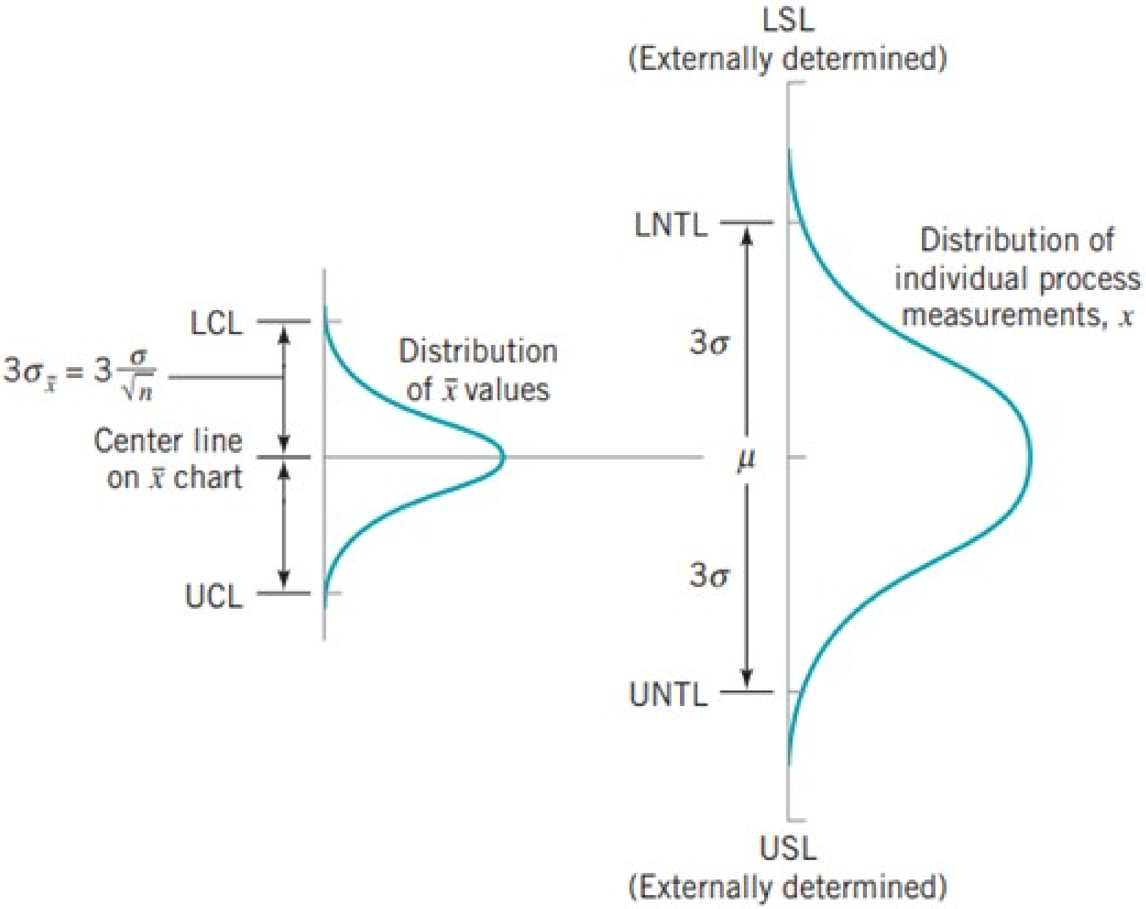
■ FIGURE 6.12 A stratification pattern.



pical control chart.

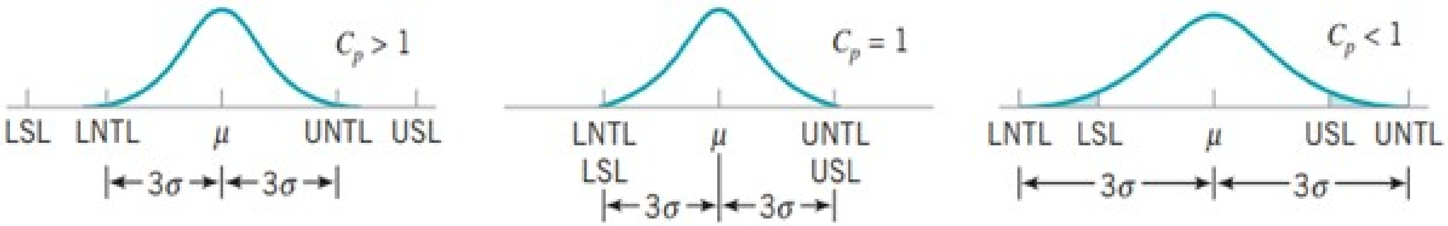
From the above chart, e.g., it appears that the process has been out of control in the 9th and 10th samples.

Control Limits, Specification Limits and Natural Tolerance Limis. A point that should be emphasized is that there is no connection or relationship between the control limits on the x and $\bar{R}$ charts and the specification limits on the process. The control limits are driven by the natural variability of the process (measured by the process standard deviation σ), that is, by the natural tolerance limits of the process. It is customary to define the upper and lower natural tolerance limits, say UNTL and LNTL, as 3σ above and below the process mean. The specification limits, on the other hand, are determined externally. They may be set by management, the manufacturing engineers, the customer or the product developers / designers. One should have knowledge of inherent process variability when setting specifications, but remember that there is no mathematical or statistical relationship between the control limits and specification limits. The situation is summarized in the figure in next slide. We have encountered practitioners who have plotted specification limits on the x control chart. This practice is completely incorrect and should not be done.


LSL
(Externally determined)
LNTL
Distribution of individual process measurements, x
3σ
LCL
$3\sigma_{\bar{x}} = 3\frac{\sigma}{\sqrt{n}}$
Distribution of $\bar{x}$ values
Center line on $\bar{x}$ chart
μ
3σ
UCL
UNTL
USL
(Externally determined)

Relationship of natural tolerance limits, control limits and specification limits.

Following figure illustrates three cases of interest relative to the PCR C_p and process specification. In figure (a) PCR C_p is greater than unity. This means that the process uses up much less than 100% of the tolerance band. Consequently, relatively few non-conforming units will be produced by this process.



(a) (b) (c)

Process fallout and the Process Capability Ratio C_p

Figure (b) shows a process for which the PCR $C_p = 1$; ie. the process uses up all the tolerance band. For a normal distribution this would imply about 0.27% (or 2700 ppm) nonconforming units. Finally figure (c) presents the process for which the PCR $C_p < 1$; ie. the process uses up more than 100% of the tolerance band. In this case the process is very yield-sensitive and a large numberof nonconforming units will be produced. Note that all the above cases assume that the process is centered at the midpoint of the specification. In many situations this will not be the case.

Estimating Process Capability **:** The $\overline{x}$ and R charts provide information about the performance or process capability of the process. From the $\overline{x}$ chart, we may estimate the mean flow width of the resist in the hard-bake process as $\overline{\overline{x}} = 1.5056$ microns. The process standard deviation may be estimated as

$$\hat{\sigma} = \overline{R}/d_2 = 0.32521/2.326 = 0.1398 \text{ microns}$$

where the value of d_2 for samples of size 5 is found in Appendix Table VI. The specification limits on flow width are 1.50 ± 0.50 microns. The control chart data may be used to describe the capability of the process to produce wafers relative to these specifications. Assuming that flow width is a normally distributed random variable, with mean 1.5056 and standard deviation 0.1398, we may estimate the fraction of nonconforming wafers produced as

$$p = P\{x < 1.00\} + P\{x > 2.99\}$$

$$= \Phi\{(1.00 – 1.5056)/0.1398\} + 1 - \Phi\{(2.00 – 1.5056)/0.1398\}$$

$$= \Phi(-3.61660) + 1 – \Phi(3.53648) \approx 0.00015 + 1 – 0.99980 \approx 0.00035$$

This is about 0.035% [350 ppm] of the wafers produced will be outside of the specifications.

Another way to express process capability is in terms of the process capability ratio (PCR) C_p, which for a quality characteristic with both upper & lower specification limits (USL & LSL respectively) is

$C_p = (USL – LSL)/6\sigma$

Note that the 6σ spread of the process is the basic definition of process capability. Since σ is usually unknown, we must replace it with an estimate. We frequently use

$\hat{\sigma} = \bar{R}/d_2$ as an estimate of σ, resulting in an estimate $\hat{C}_p$ of C_p. For the hard-bake process, since $\bar{R}/d_2 = \hat{\sigma} = 0.1398$, we find that

$\hat{C}_p = (2 – 1)/6(0.1398) = 1/0.8388 = 1.192$

This implies that the “natural” tolerance limits in the process (3σ above & below the mean) are inside the lower & upper specification limits. Consequently, a moderately small number of nonconforming wafers will be produced. The PCR C_p may be interpreted another way. The quantity

$P = (1/C_p)100\%$

is simply the percentage of the specification band that the process uses up. For the hard-bake process an estimate of P is

$\hat{P} = (1/C_{\hat{p}})100\% = (1/1.192)100\% = 83.89$

That is, the process uses up about 84% of the specification band.

Charts based on Standard Values **:** When it is possible to specify standard values for the process mean & standard deviation, we may use these standards to establish the control charts for $\bar{x}$ and R without analysis of past data. Suppose that the standards given are μ and σ. Then the parameters of $\bar{x}$ chart are

UCL = $\mu + 3\sigma/\sqrt{n}$

Center line = μ

LCL = $\mu - 3\sigma/\sqrt{n}$

Thus the parameters of the R chart with standard σ given are

UCL = $D_2\sigma$

Center line = $d_2\sigma$

LCL = $D_1\sigma$

Control Charts for $\bar{x}$ and s **:** Although $\bar{x}$ and R charts are widely used, it is occasionally desirable to estimate the process standard deviation directly instead of indirectly through the use of the Range R. This leads to control charts for $\bar{x}$ and s, where s is the sample standard deviation. Generally, $\bar{x}$ and s charts are preferable to their more familiar counterparts, $\bar{x}$ and R charts when either

1. The sample size n is moderately large say, $n > 10$ or 12. (Recall that the range method for estimating σ loses statistical efficiency for moderate to large samples).
2. The sample size n is variable.

Consequently, the parameters of the s chart with a standard value for σ given become

$UCL = B_6\sigma$

$\text{Center line} = C_4\sigma$

$LCL = B_5\sigma$

Values of B_5 & B_6 are tabulated for various sample sizes in Appendix Table VI. The parameters of the corresponding $\bar{x}$ chart have been given earlier.

If no standard is given for σ, then it must be estimated by analyzing past data. Suppose that m preliminary samples are available, each of size n, and let s_i be the standard deviation of the i^{th} sample. The average of the m standard deviation is

$$\bar{s} = (1/m)\left(\sum_{i=1}^{m} s_i\right)$$

$$s_i = \sqrt{\left\{\sum_{i=1}^{n} (x_i - \bar{x})^2/(n-1)\right\}}$$

Consequently, we may write the parameters of the s chart as

$UCL = B_4\bar{s}$

$\text{Center line} = \bar{s}$

$LCL = B_3\bar{s}$

Let the constant $A_3 = 3/(C_4\sqrt{n})$. Then the $\bar{x}$ chart parameters become

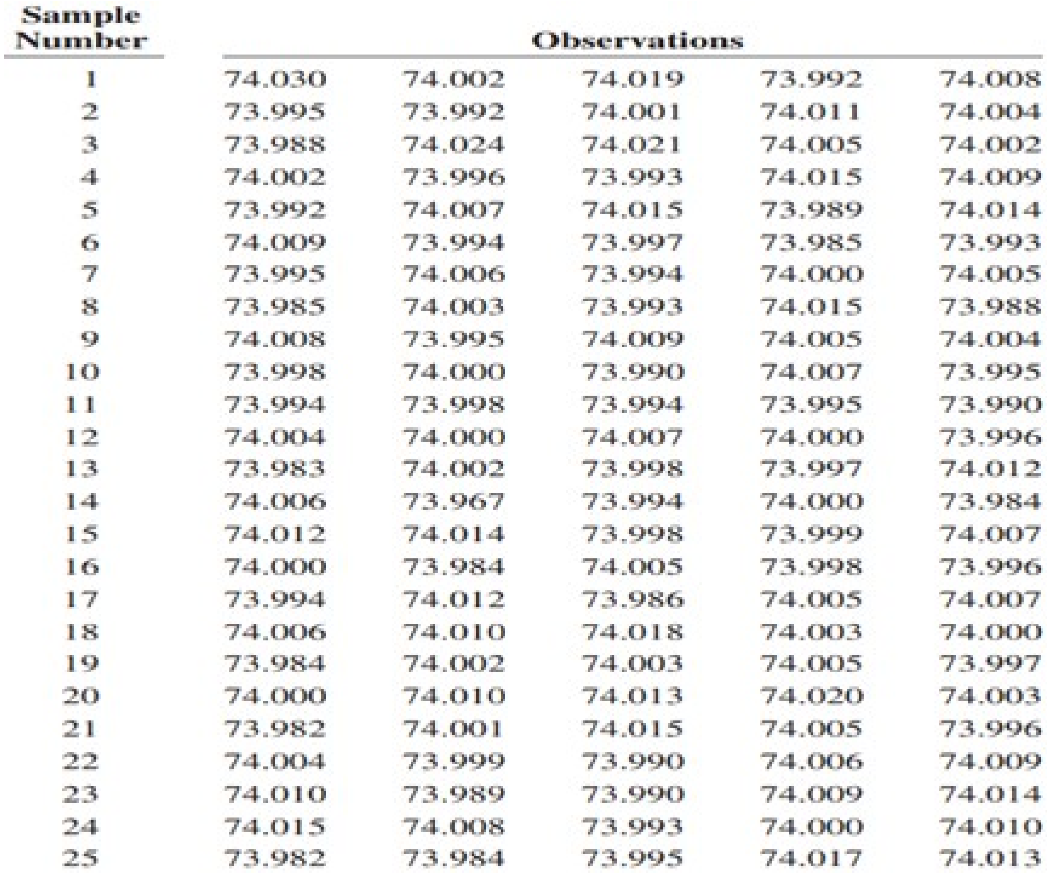
$UCL = \bar{\bar{x}} + A_3\bar{s}$

$\text{Center line} = \bar{\bar{x}}$

$LCL = \bar{\bar{x}} - A_3\bar{s}$

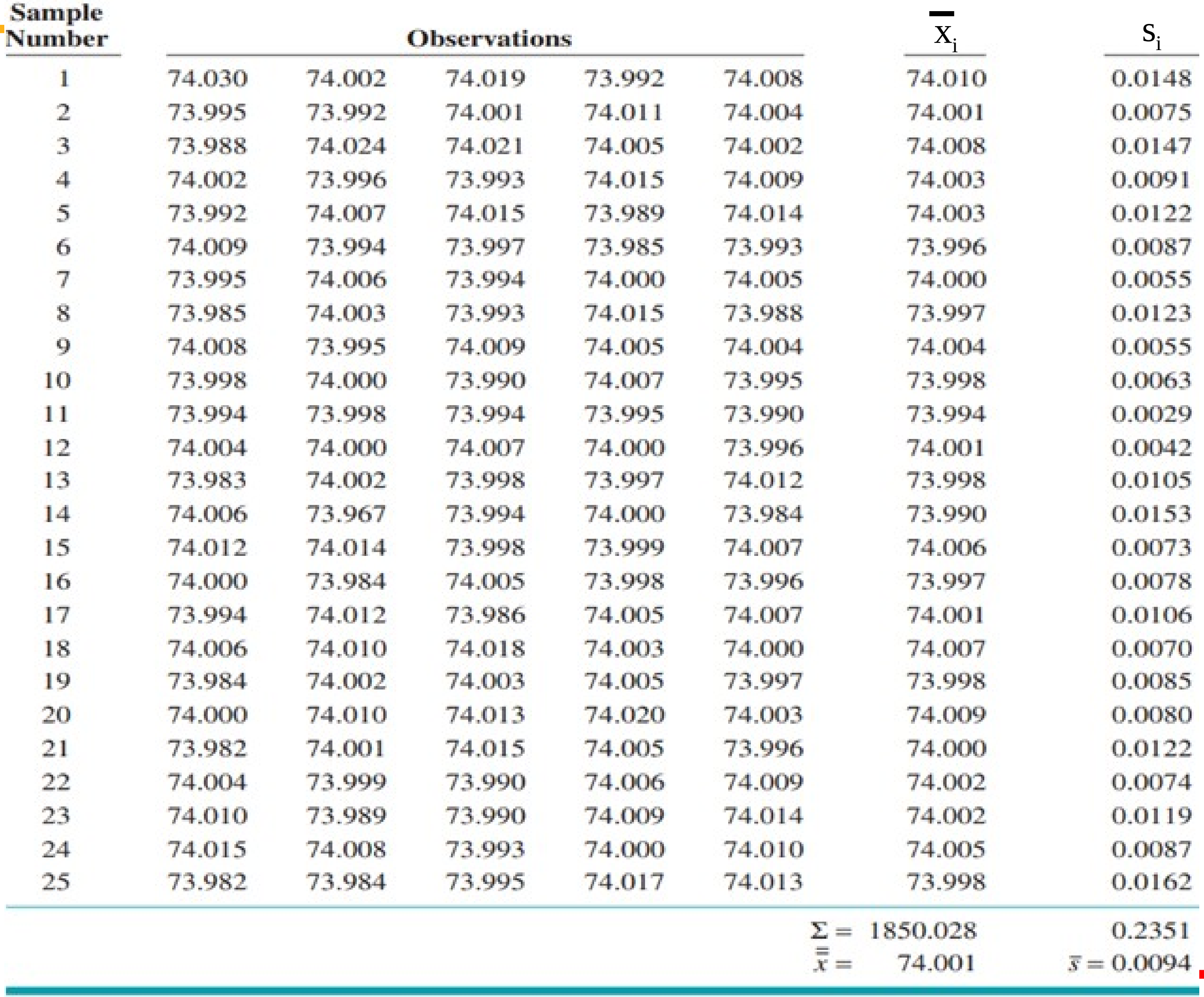
Ex : $\bar{x}$ and s charts for the Piston Ring Data – Construct & interpret x & s charts using the piston ring inside diameter measurements in the following table.

Inside Diameter Measurements (mm) for Automobile Engine Piston Ring



Sample Number	Observations				
1	74.030	74.002	74.019	73.992	74.008
2	73.995	73.992	74.001	74.011	74.004
3	73.988	74.024	74.021	74.005	74.002
4	74.002	73.996	73.993	74.015	74.009
5	73.992	74.007	74.015	73.989	74.014
6	74.009	73.994	73.997	73.985	73.993
7	73.995	74.006	73.994	74.000	74.005
8	73.985	74.003	73.993	74.015	73.988
9	74.008	73.995	74.009	74.005	74.004
10	73.998	74.000	73.990	74.007	73.995
11	73.994	73.998	73.994	73.995	73.990
12	74.004	74.000	74.007	74.000	73.996
13	73.983	74.002	73.998	73.997	74.012
14	74.006	73.967	73.994	74.000	73.984
15	74.012	74.014	73.998	73.999	74.007
16	74.000	73.984	74.005	73.998	73.996
17	73.994	74.012	73.986	74.005	74.007
18	74.006	74.010	74.018	74.003	74.000
19	73.984	74.002	74.003	74.005	73.997
20	74.000	74.010	74.013	74.020	74.003
21	73.982	74.001	74.015	74.005	73.996
22	74.004	73.999	73.990	74.006	74.009
23	74.010	73.989	73.990	74.009	74.014
24	74.015	74.008	73.993	74.000	74.010
25	73.982	73.984	73.995	74.017	74.013

Soln :



Sample Number	Observations					$\bar{x}_i$	s_i
1	74.030	74.002	74.019	73.992	74.008	74.010	0.0148
2	73.995	73.992	74.001	74.011	74.004	74.001	0.0075
3	73.988	74.024	74.021	74.005	74.002	74.008	0.0147
4	74.002	73.996	73.993	74.015	74.009	74.003	0.0091
5	73.992	74.007	74.015	73.989	74.014	74.003	0.0122
6	74.009	73.994	73.997	73.985	73.993	73.996	0.0087
7	73.995	74.006	73.994	74.000	74.005	74.000	0.0055
8	73.985	74.003	73.993	74.015	73.988	73.997	0.0123
9	74.008	73.995	74.009	74.005	74.004	74.004	0.0055
10	73.998	74.000	73.990	74.007	73.995	73.998	0.0063
11	73.994	73.998	73.994	73.995	73.990	73.994	0.0029
12	74.004	74.000	74.007	74.000	73.996	74.001	0.0042
13	73.983	74.002	73.998	73.997	74.012	73.998	0.0105
14	74.006	73.967	73.994	74.000	73.984	73.990	0.0153
15	74.012	74.014	73.998	73.999	74.007	74.006	0.0073
16	74.000	73.984	74.005	73.998	73.996	73.997	0.0078
17	73.994	74.012	73.986	74.005	74.007	74.001	0.0106
18	74.006	74.010	74.018	74.003	74.000	74.007	0.0070
19	73.984	74.002	74.003	74.005	73.997	73.998	0.0085
20	74.000	74.010	74.013	74.020	74.003	74.009	0.0080
21	73.982	74.001	74.015	74.005	73.996	74.000	0.0122
22	74.004	73.999	73.990	74.006	74.009	74.002	0.0074
23	74.010	73.989	73.990	74.009	74.014	74.002	0.0119
24	74.015	74.008	73.993	74.000	74.010	74.005	0.0087
25	73.982	73.984	73.995	74.017	74.013	73.998	0.0162
					$\Sigma =$	1850.028	0.2351
					$\bar{\bar{x}} =$	74.001	$\bar{s} =$ 0.0094

The grand average and the average standard deviation are

$$\bar{\bar{x}} = (1/25)\sum_{i=1}^{25} \bar{x}_i = (1/25)(1850.028) = 74.001 \quad \text{and}$$

$$\bar{s} = (1/25)\sum_{i=1}^{25} s_i = (1/25)(0.2351) = 0.0094$$

respectively. Consequently, the parameters for the $\bar{x}$ chart are

$$UCL = \bar{\bar{x}} + A_3\bar{s} = 74.001 + (1.427)(0.0094) = 74.014$$

$$CL = \bar{\bar{x}} = 74.001$$

$$LCL = \bar{\bar{x}} - A_3\bar{s} = 74.001 - (1.427)(0.0094) = 73.988$$

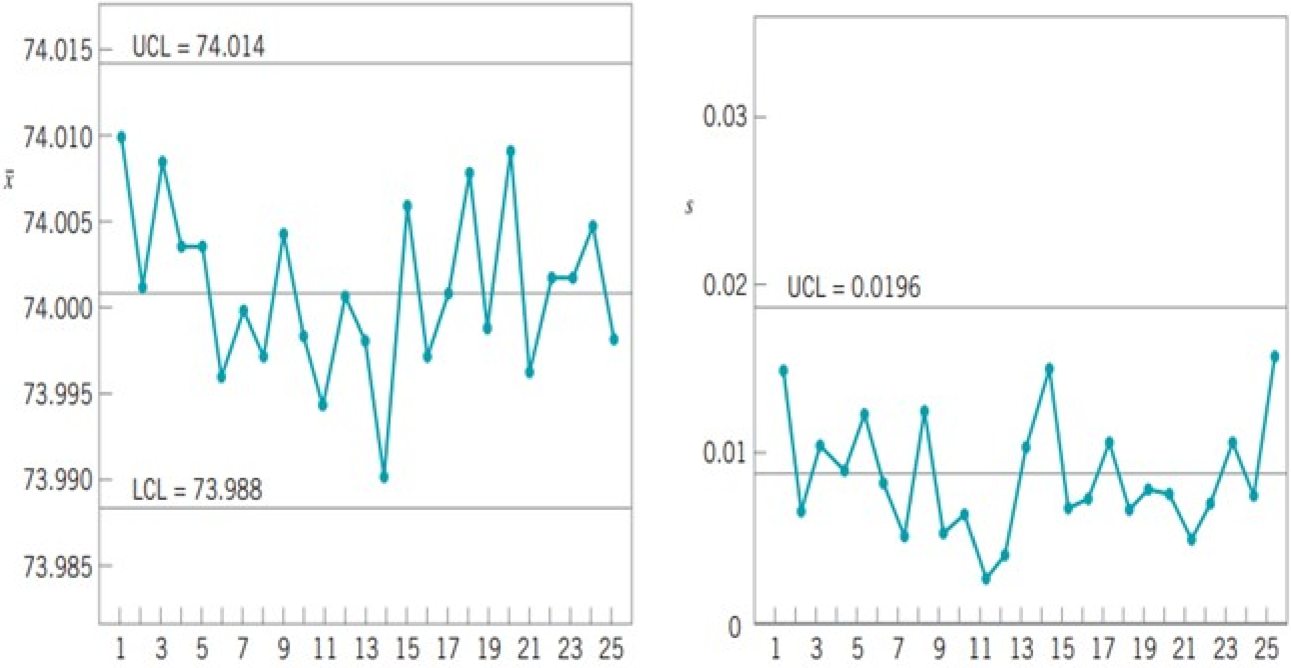
and the s chart

$$UCL = B_4\bar{s} = (2.089)(0.0094) = 0.0196$$

$$Cl = \bar{s} = 0.0094$$

$$LCL = B_3s = (0)(0.0094) = 0$$

The control charts are shown here. There is no indication that the process is out of control, so those limits could be adopted for phase II monitoring of the process.



Sample number

(a)

Sample number

(b)

(a) $\bar{x}$ chart and (b) s chart for the given example

Control Chart of Attribute

p-Chart

np-Chart

C-Chart

Control Chart for Fraction Nonconforming or Defective (p)

The fraction nonconforming or defective (p) is defined as the ratio of the number of nonconforming items in a population to the total number of items in that population. The items may have several quality characteristics that are examined simultaneously by the inspector. If the item does not conform to standard on one or more of these characteristics, it is classified as nonconforming.

The statistical principles underlying the control chart for fraction nonconforming are based on the binomial distribution. Suppose the production process is operating in a stable manner, such that probability that any unit will not conform to specifications is p and successive units produced are independent. Then each unit produced is a realization of Bernoulli random variable with parameter p. If the random sample of n units of product is selected and if D is the number of units of product that are nonconforming then

Control Chart for Fraction Nonconforming or Defective (p)

$P(D = x) = {}^{n}C_{x}p^{x}(1 - p)^{n-x}$, for $x = 0, 1, \ldots, n$

We know for binomial random variable D mean & variance are np & np(1 – p) respectively.

The sample fraction nonconforming is defined as the ratio of the number of nonconforming units in the sample D to the sample size n; that is

$\hat{p} = D/n$

Suppose that the true fraction nonconforming p in the production process is known or is specified standard value. Then of the center line & control limits of the fraction nonconforming control chart would be as follows –

Fraction Nonconforming Control Chart : Standard Given

$UCL = p + 3\sqrt{\{p(1 - p)/n\}}$

Center line = p

$LCL = p - 3\sqrt{\{p(1 - p)/n\}}$

Depending on the values of p and n, sometimes the lower control limit $LCL < 0$. In these cases, we customarily set $LCL = 0$ and assume that the control chart only has an upper control limit.

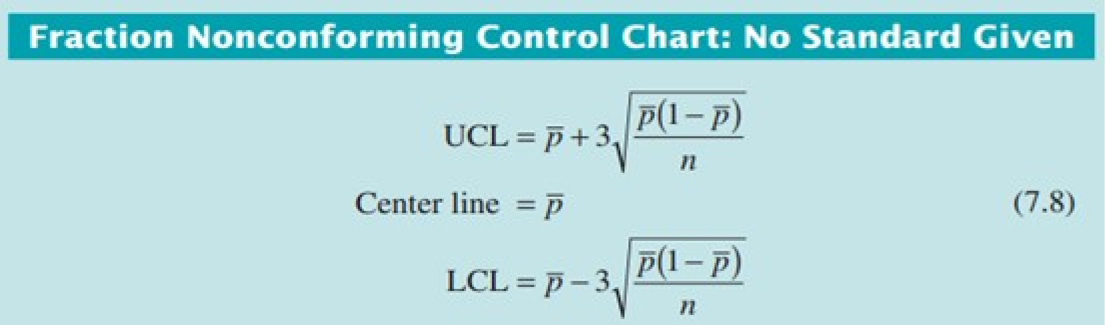
When the process fraction nonconforming p is not known, then it must be estimated from observed data. The usual procedure is to select m preliminary samples, each of size n. As the general rule, m should be at least 20 or 25. Then if there are D_i nonconforming units in sample i, we compute the fraction nonconforming in the $i^{\underline{th}}$ sample as

$$\hat{p}_i = D_i/n, \quad i = 1, 2, \ldots, m$$

and the average of these individual samples fractions nonconforming is

$$\bar{p} = \{(\sum_{i=1}^{m} D_i)/mn\} = \{(\sum_{i=1}^{m} \hat{p}_i)/m\}$$

The statistic $\bar{p}$ estimates the unknown fraction nonconforming p. The center line and control limits of the control chart for fraction nonconforming are computed as given in next slide.



Fraction Nonconforming Control Chart: No Standard Given

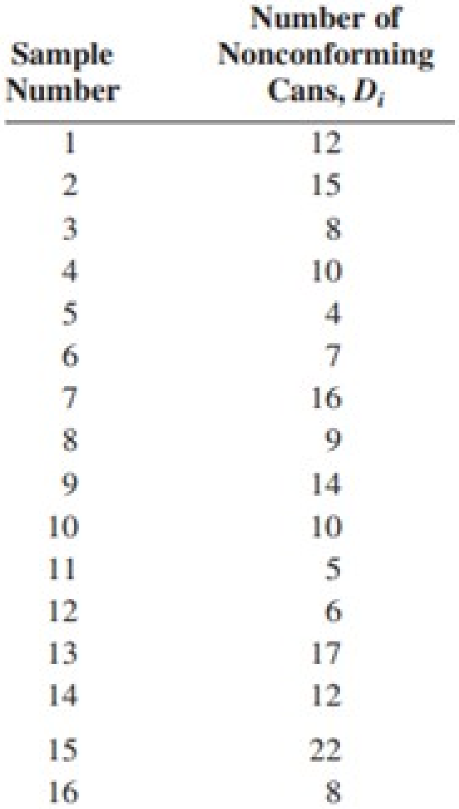
$$\begin{aligned}\text{UCL} &= \bar{p} + 3\sqrt{\frac{\bar{p}(1-\bar{p})}{n}}\\ \text{Center line} &= \bar{p} \qquad\qquad (7.8)\\ \text{LCL} &= \bar{p} - 3\sqrt{\frac{\bar{p}(1-\bar{p})}{n}}\end{aligned}$$

This control chart is also often called the p-chart.

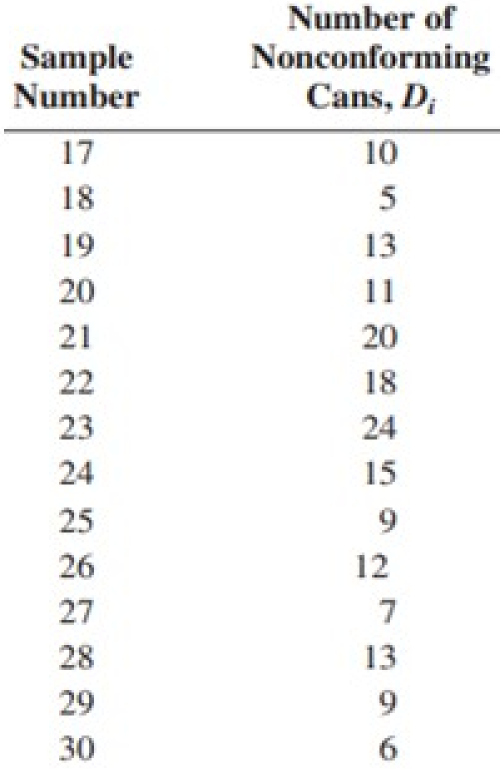
Ex : Construction & operation of a Fraction Nonconforming Control Chart

Frozen orange juice concentrate is packed in 6-oz cardboard cans. These cans are formed on a machine by spinning them from cardboard stock and attaching a metal bottom panel. By inspection of a can, we may determine whether, when filled, it could possibly leak either on the side seam or around the bottom joint. Such a nonconforming can has an improper seal on either the side seam or the bottom panel. Set up a control chart to improve the fraction of nonconforming cans produced by this machine.

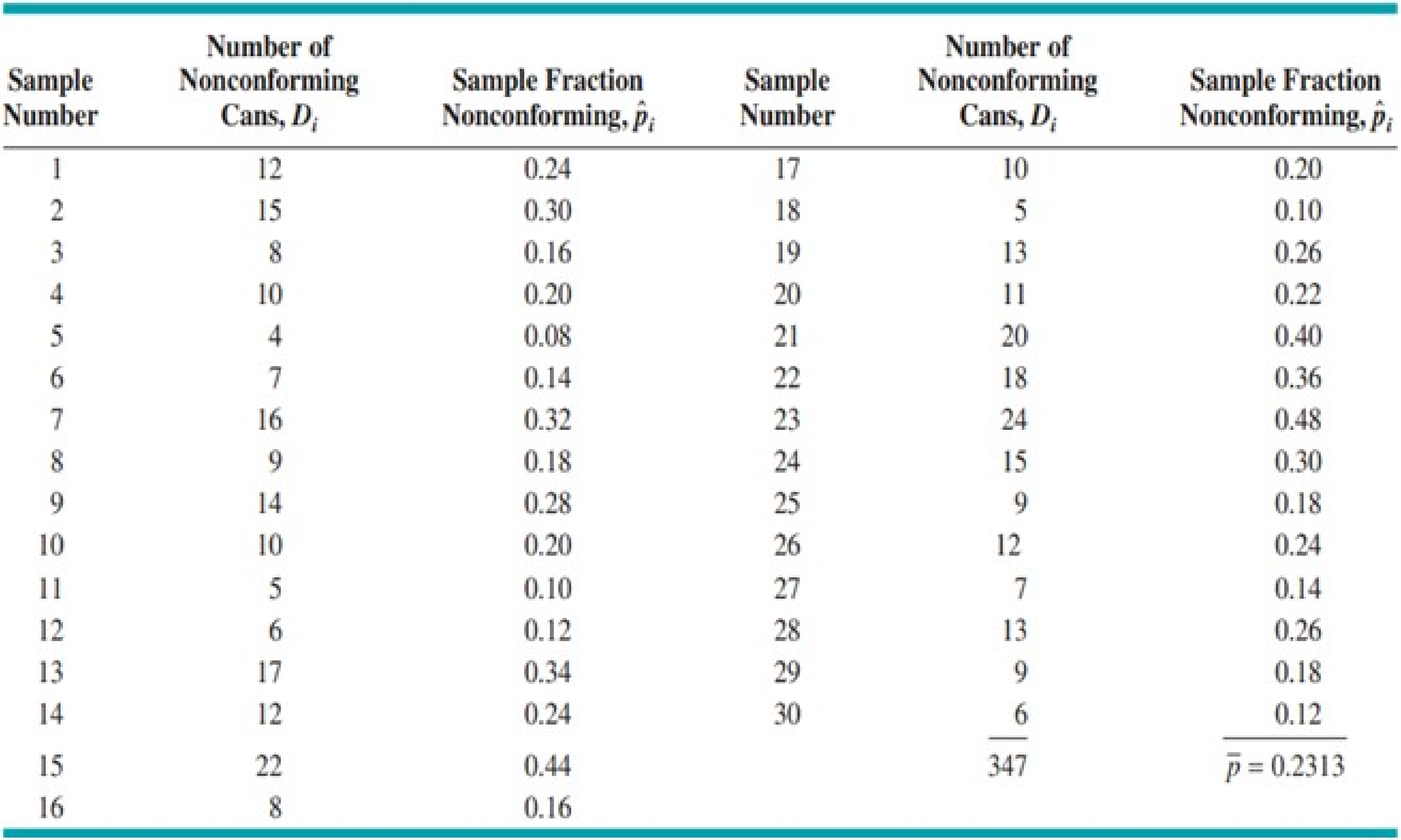
Data for Trial Control Limits, Sample size = 50



Sample Number	Number of Nonconforming Cans, D_i
1	12
2	15
3	8
4	10
5	4
6	7
7	16
8	9
9	14
10	10
11	5
12	6
13	17
14	12
15	22
16	8



Sample Number	Number of Nonconforming Cans, D_i
17	10
18	5
19	13
20	11
21	20
22	18
23	24
24	15
25	9
26	12
27	7
28	13
29	9
30	6



Sample Number	Number of Nonconforming Cans, D_i	Sample Fraction Nonconforming, $\hat{p}_i$	Sample Number	Number of Nonconforming Cans, D_i	Sample Fraction Nonconforming, $\hat{p}_i$
1	12	0.24	17	10	0.20
2	15	0.30	18	5	0.10
3	8	0.16	19	13	0.26
4	10	0.20	20	11	0.22
5	4	0.08	21	20	0.40
6	7	0.14	22	18	0.36
7	16	0.32	23	24	0.48
8	9	0.18	24	15	0.30
9	14	0.28	25	9	0.18
10	10	0.20	26	12	0.24
11	5	0.10	27	7	0.14
12	6	0.12	28	13	0.26
13	17	0.34	29	9	0.18
14	12	0.24	30	6	0.12
15	22	0.44		347	$\bar{p} = 0.2313$
16	8	0.16			

To establish the control chart, 30 samples of sample size, n = 50 cans each ere selected at half-hour intervals over a three-shift period in which the machine was in continuous operation. The data are shown in the table that follows.

We construct a phase I control chart using the preliminary data to determine if the process was in control when these data were collected. Since the 30 samples contain $\sum_{i=1}^{30} D_i = 347$ nonconforming cans, find

$$\bar{p} = \sum_{i=1}^{30} D_i/mn = 347/30x50 = 0.2313$$

Using p as an estimate of the true process fraction nonconforming, we can now calculate the upper & lower control limits as

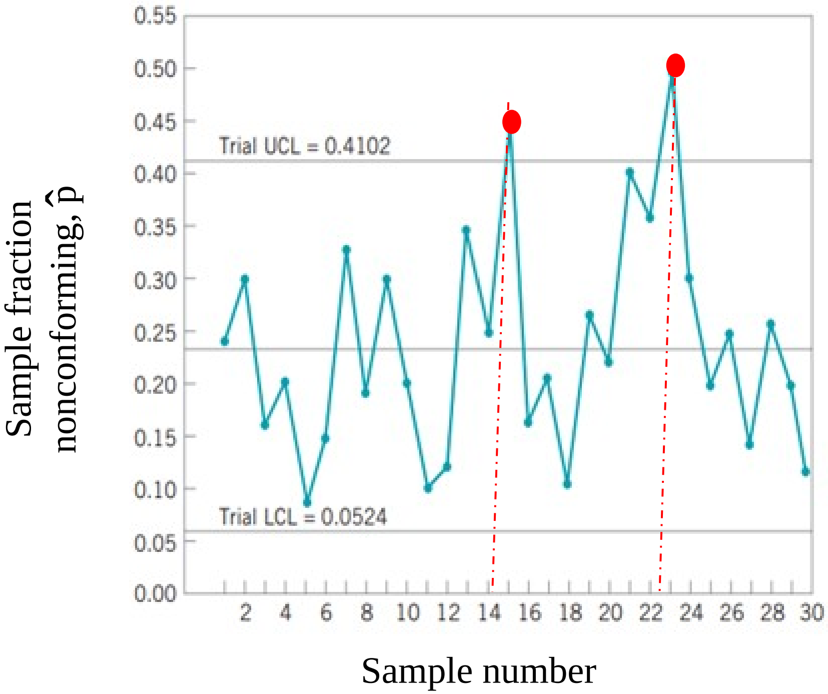
$$\bar{p} \pm 3\sqrt{\{\bar{p}(1 - \bar{p})/n\}} = 0.2313 \pm 3\sqrt{\{0.2313(0.7687)/50\}} = 0.2313 \pm 0.1789$$

Therefore,

$$UCL = \bar{p} + 3\sqrt{\{\bar{p}(1 - \bar{p})/n\}} = 0.2313 + 0.1789 = 0.4102$$

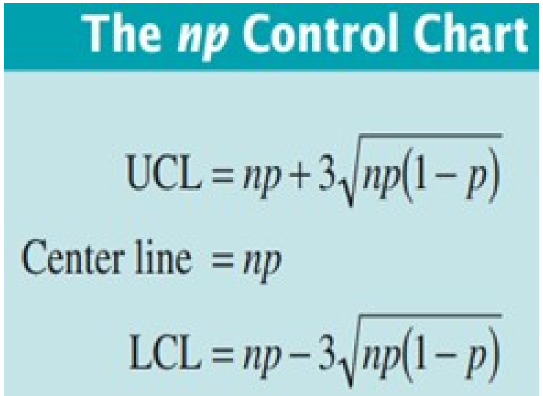
$$LCL = \bar{p} - 3\sqrt{\{\bar{p}(1 - \bar{p})/n\}} = 0.2313 - 0.1789 = 0.0524$$

The control chart with center line at $\overline{p} = 0.2313$ and the above UCL & LCL are shown in the figure. The sample fraction nonconforming from each preliminary sample is plotted on this chart. We need that two points, those from sample 15 & 23, plot above the UCL, so the process is not in control. These points must be investigated to see whether an assignable cause can be determined.



Initial phase I fraction nonconforming control chart for given data

The np Control Chart **:** It is possible to base a control chart on the number non-conforming rather than the fraction nonconforming. This is often called an number nonconforming (np) control chart. The parameters of this chart are as follows.



The *np* Control Chart

$$\text{UCL} = np + 3\sqrt{np(1-p)}$$

$$\text{Center line} = np$$

$$\text{LCL} = np - 3\sqrt{np(1-p)}$$

If a standard value for p is unavailable, then $\bar{p}$ can be used to estimate p. Many nonstatistically trained personnel find the np chart easier to interpret than the usual fraction nonconforming control chart.

Ex : An np Control Chart - Set up an np control chart for the orange juice concentrate can process in earlier example.

Using the data in the table, we found that $\bar{p} = 0.2313$, $n = 50$

Soln : The parameters of the np control chart would be

$UCL = n\bar{p} + 3\sqrt{\{n\bar{p}(1 - \bar{p})\}} = 50(0.2313) + 3\sqrt{\{50)(0.2313)(0.7687)\}} = 20.510$

$CL = n\bar{p} = 50(0.2310) = 11.565$

$LCL = n\bar{p} - 3\sqrt{\{n\bar{p}(1 - \bar{p})\}} = 50(0.2313) - 3\sqrt{\{50)(0.2313)(0.7687)\}} = 2.620$

Now in practice, the number of nonconforming units in each sample is plotted on the np control chart, and the number of nonconforming units is an integer. Thus, if 20 units are nonconforming the process is in control, but if 21 occur the process is out of control. Similarly, there are three nonconforming units in the sample and the process is in control, but two nonconforming units would imply an out-of-control process. Some practitioners prefer to use integer values for the control limits on the np chart instead of their decimal fraction counterparts. In this example we could choose 2 and 21 as the LCL and UCL respectiverly and the process would be considered out-of-control if a sample value of np plotted at or beyond the control limits.

Variable Sample Size **:** In some applications of the control chart for fraction nonconforming, the sample is a 100% inspection of process output over some period of time. Since different numbers of units could be produced in each period, the control chart would then have a variable sample size. There

Variable-Width Control Limits **:** The first and perhaps the most simple approach is to determine control limits for each individual sample that are based on the specific sample size. That is, if the i^{th} sample is of size n_i, then the upper and lower control limits are $\bar{p} \pm 3\surd\{\bar{p}(1 - \bar{p})/ni$. Note that the width of the control limits is inversely proportional to the square root of the sample size.

To illustrate this approach, consider the data in Table 7.4. These data came from the purchasing group of a large aerospace company. This group issues purchase orders to the company’s suppliers. The sample size in Table 7.4 are the total number of purchase orders issued each week. Obviously, this is not constant. A nonconforming unit is a purchase order with an error. Among the most common errors are specifying incorrect part numbers, wrong delivery dates and wrong supplier information. Any of these mistakes can result in a purchase order change, which takes time and resources and may result in delayed delivery of material.

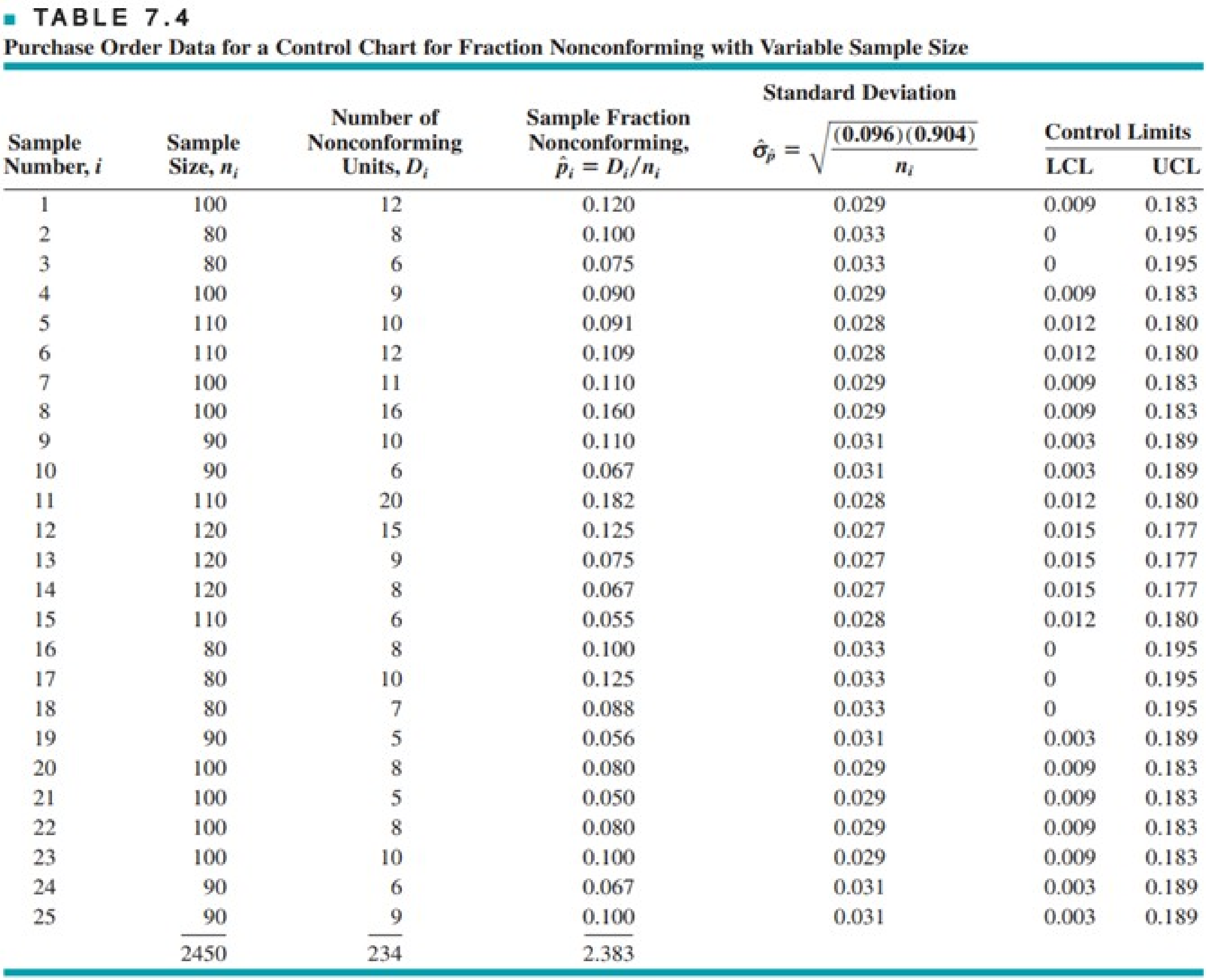


TABLE 7.4

Purchase Order Data for a Control Chart for Fraction Nonconforming with Variable Sample Size

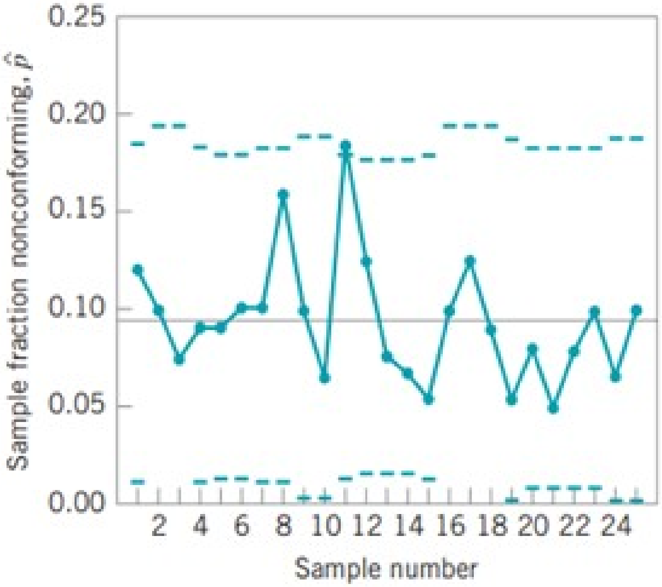
Sample Number, i	Sample Size, n_i	Number of Nonconforming Units, D_i	Sample Fraction Nonconforming, $\hat{p}_i = D_i/n_i$	Standard Deviation $\hat{\sigma}_{\hat{p}} = \sqrt{\frac{(0.096)(0.904)}{n_i}}$	Control Limits LCL	Control Limits UCL
1	100	12	0.120	0.029	0.009	0.183
2	80	8	0.100	0.033	0	0.195
3	80	6	0.075	0.033	0	0.195
4	100	9	0.090	0.029	0.009	0.183
5	110	10	0.091	0.028	0.012	0.180
6	110	12	0.109	0.028	0.012	0.180
7	100	11	0.110	0.029	0.009	0.183
8	100	16	0.160	0.029	0.009	0.183
9	90	10	0.110	0.031	0.003	0.189
10	90	6	0.067	0.031	0.003	0.189
11	110	20	0.182	0.028	0.012	0.180
12	120	15	0.125	0.027	0.015	0.177
13	120	9	0.075	0.027	0.015	0.177
14	120	8	0.067	0.027	0.015	0.177
15	110	6	0.055	0.028	0.012	0.180
16	80	8	0.100	0.033	0	0.195
17	80	10	0.125	0.033	0	0.195
18	80	7	0.088	0.033	0	0.195
19	90	5	0.056	0.031	0.003	0.189
20	100	8	0.080	0.029	0.009	0.183
21	100	5	0.050	0.029	0.009	0.183
22	100	8	0.080	0.029	0.009	0.183
23	100	10	0.100	0.029	0.009	0.183
24	90	6	0.067	0.031	0.003	0.189
25	90	9	0.100	0.031	0.003	0.189
	2450	234	2.383			

For 25 samples, we calculate $\bar{p} = \sum_{i=1}^{25} D_i / \sum_{i=1}^{25} n_i = 234/2450 = 0.096$

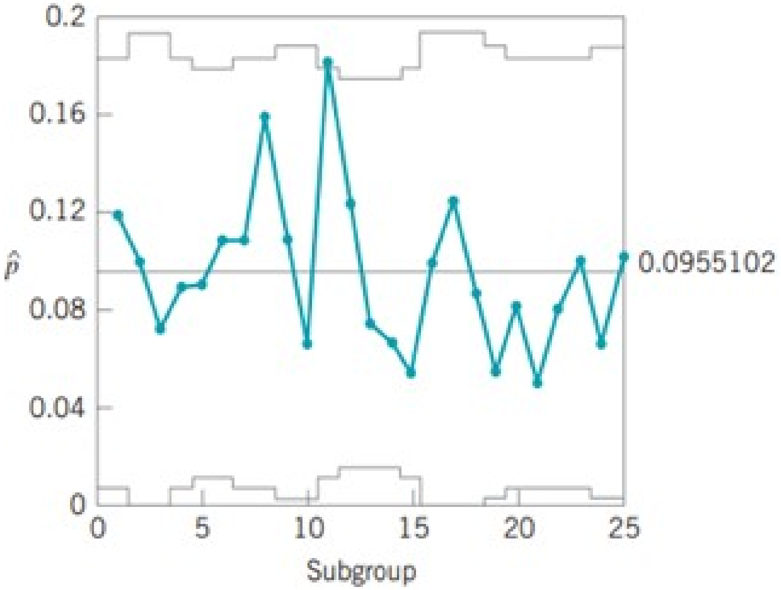
Consequently, the center line is at 0.096 and control limits are

$UCL = \bar{p} + 3\hat{\sigma}_{\hat{p}} = 0.096 + 3\sqrt{(0.096)(0.904)/n_i}$

$LCL = \bar{p} - 3\hat{\sigma}_{\hat{p}} = 0.096 - 3\sqrt{(0.096)(0.904)/n_i}$



Control chart for fraction nonconforming with variable sample size



Control chart for fraction nonconforming with variable sample size using Minitab

C-Control Chart for Nonconformities (Number of Defects per item:C)

A nonconforming item is a unit of product that does not satisfy one or more of the specifications for that product. Each specific point at which a specification is not satisfied results in a defect or nonconformity. Consequently, a nonconforming item will contain at least one nonconformity. However, depending on their nature and severity, it is quite possible for a unit to contain several nonconformities and not be classified as nonconforming. As an example, suppose we are manufacturing personal computers. Each unit could have one or more very minor flaws in the cabinet finish, and since these flaws do not seriously affect the unit’s functional operation, it could be classified as conforming. However, if there are too many of these flaws, the personal computer should be classified as nonconforming, since the flaws will be very noticeable to the customer and might affect the sale of the unit. There are many practical situations in which we prefer to work directly with the number of defects or nonconformities rather than the fraction nonconforming. These include the number of defective welds in 100 m of oil pipeline, the number of broken reivets in an aircraft wing, the number of functional defects in an electronic logic device, the number of errors on a document and so forth.

Consider the occurrence of nonconformities in an inspection unit of product. In most cases, the inspection unit will be a single unit of product, although this is not necessarily always so. The inspection unit is simply an entity for which it is convenient to keep records. It could be a group of 5 units of product, 10 units of product and so on. Suppose that defects or nonconformities occur in this inspection unit according to the Poisson distribution; that is

$p(x) = e^{-c}c^{x}/x! \quad x = 0, 1, 2, \ldots\ldots$

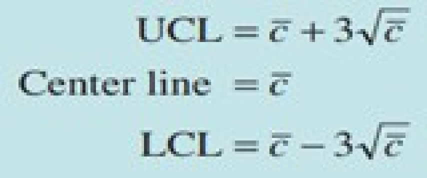
where x is the number of nonconformities and $c > 0$ is the parameter of the Poisson distribution. We know that the mean and variance of the Poisson distribution are the parameter c. Therefore, a control chart for nonconformities, or c chart with three sigma limits would be defined as follows.

Control Chart for Nonconformities : Standard Given

$$\begin{aligned} \text{UCL} &= c + 3\sqrt{c} \\ \text{Center line} &= c \\ \text{LCL} &= c - 3\sqrt{c} \end{aligned}$$

Assuming that a standard value for c is available. Should these calculations yield a negative value for the LCL, set LCL = 0

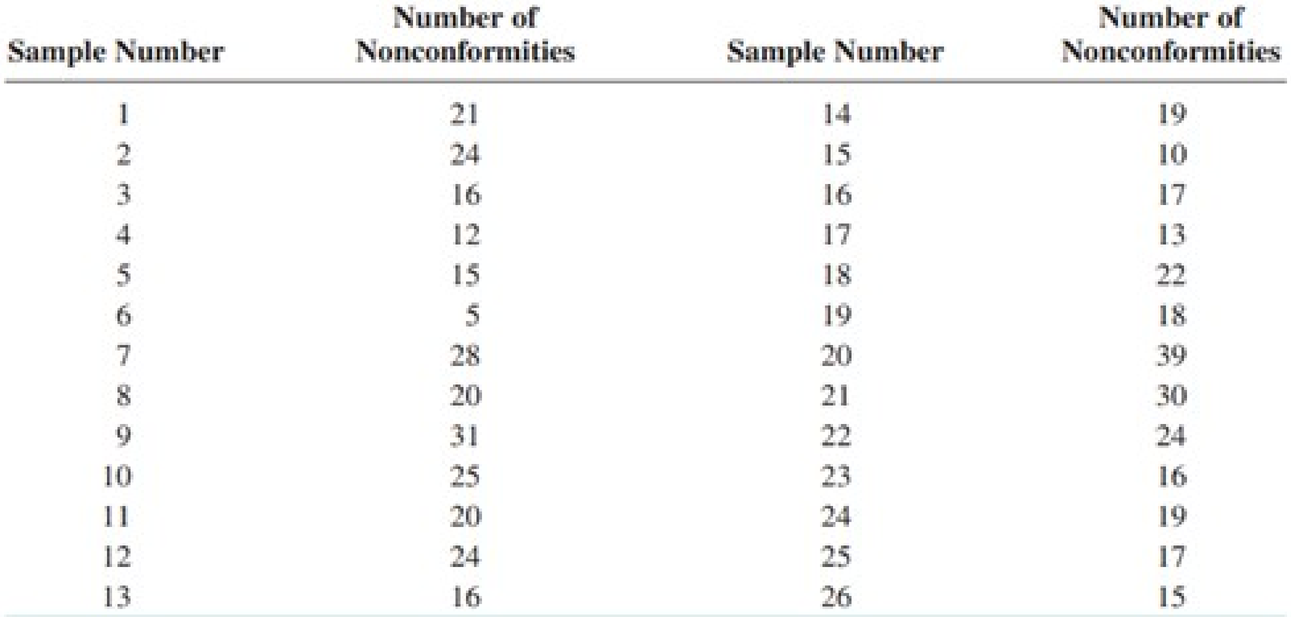
Control chart for Nonconformities **:** No Standard Given



$$\begin{aligned} \text{UCL} &= \bar{c} + 3\sqrt{\bar{c}} \\ \text{Center line} &= \bar{c} \\ \text{LCL} &= \bar{c} - 3\sqrt{\bar{c}} \end{aligned}$$

Ex : Nonconformities in Printed Circuit Boards - Table 7.7 below presents the number of nonconformities observed in 26 successive samples of 100 PCBs. Note that, for reasons of convenience, the inspection unit is defined as 100 boards. Set up a c chart for these data.

Table 7.7 Data on the Number of Nonconformities in Samples of 100 PCBs



Sample Number	Number of Nonconformities	Sample Number	Number of Nonconformities
1	21	14	19
2	24	15	10
3	16	16	17
4	12	17	13
5	15	18	22
6	5	19	18
7	28	20	39
8	20	21	30
9	31	22	24
10	25	23	16
11	20	24	19
12	24	25	17
13	16	26	15

Soln : Since 26 samples contain 516 total nonconformities, we estimate c by

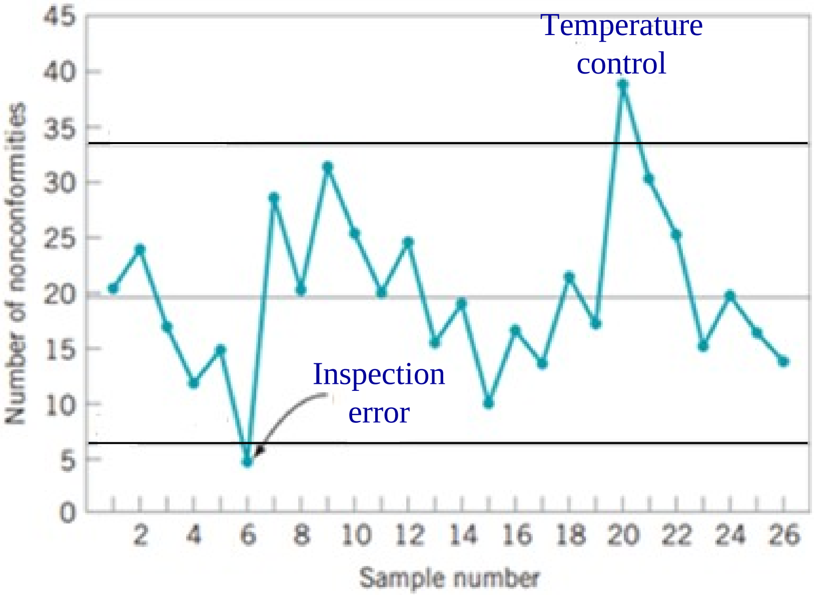
$\bar{c} = 516/26 = 19.85$

Therefore, the trial control limits are given by

$UCL = \bar{c} + 3\sqrt{\bar{c}} = 19.85 + 3\sqrt{19.85} = 33.22$

$CL = \bar{c} = 19.85$

$LCL = \bar{c} - 3\sqrt{\bar{c}} = 19.85 - 3\sqrt{19.85} = 6.48$



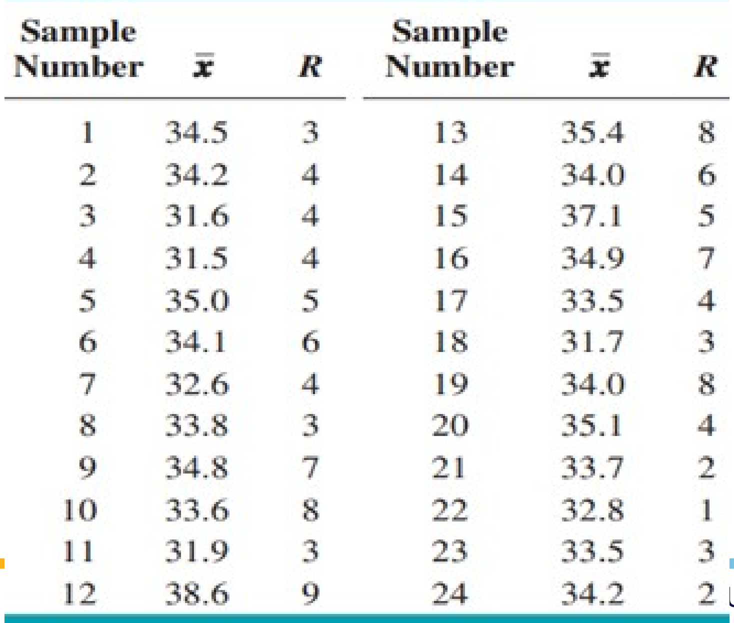
The control chart for the nonconformities is shown here. The number of observed nonconformities from the preliminary samples is plotted on this chart. Two points (samples 6 & 20) plot outside the control limits. Investigation of sample 6 revealed that a new inspector had examined the PCBs in this sample & that he did not recognize several of the types of nonconformities that could have been present. Furthermore, the unusually large number of nonconformities in sample 20 resulted from a temperature control problem in the wave soldering machine, which was subsequently repaired.

Ex : The data shown in the table are x and R values for 24 samples of size n = 5 taken from a process producing bearings. The measurements are made on the inside diameter of the bearing, with only last three decimals recorded (ie., 34.5 should be recorded 0.50345).

(a) Set up $\bar{x}$ and R charts on this process. Does the process seem to be in statistical control?

(b) If specifications on this diameter are 0.5030 ± 0.0010, find the percentage of nonconforming bearings produced by this process. Assume that diameter is normally distributed.

Bearing Diameter Data



Sample Number	$\bar{x}$	R	Sample Number	$\bar{x}$	R
1	34.5	3	13	35.4	8
2	34.2	4	14	34.0	6
3	31.6	4	15	37.1	5
4	31.5	4	16	34.9	7
5	35.0	5	17	33.5	4
6	34.1	6	18	31.7	3
7	32.6	4	19	34.0	8
8	33.8	3	20	35.1	4
9	34.8	7	21	33.7	2
10	33.6	8	22	32.8	1
11	31.9	3	23	33.5	3
12	38.6	9	24	34.2	2

Samples of n = 6 items each are taken from a process at regular intervals. A quality characteristic is measured, and $\bar{x}$ and R values are calculated for each sample. After 50 samples, we have

$$\sum_{i=1}^{50} \bar{x}_i = 2000 \quad \text{and} \quad \sum_{i=1}^{50} R_i = 200$$

Assume that the quality characteristic is normally distributed.

(a) Compute control limits for the $\bar{x}$ and R control charts.

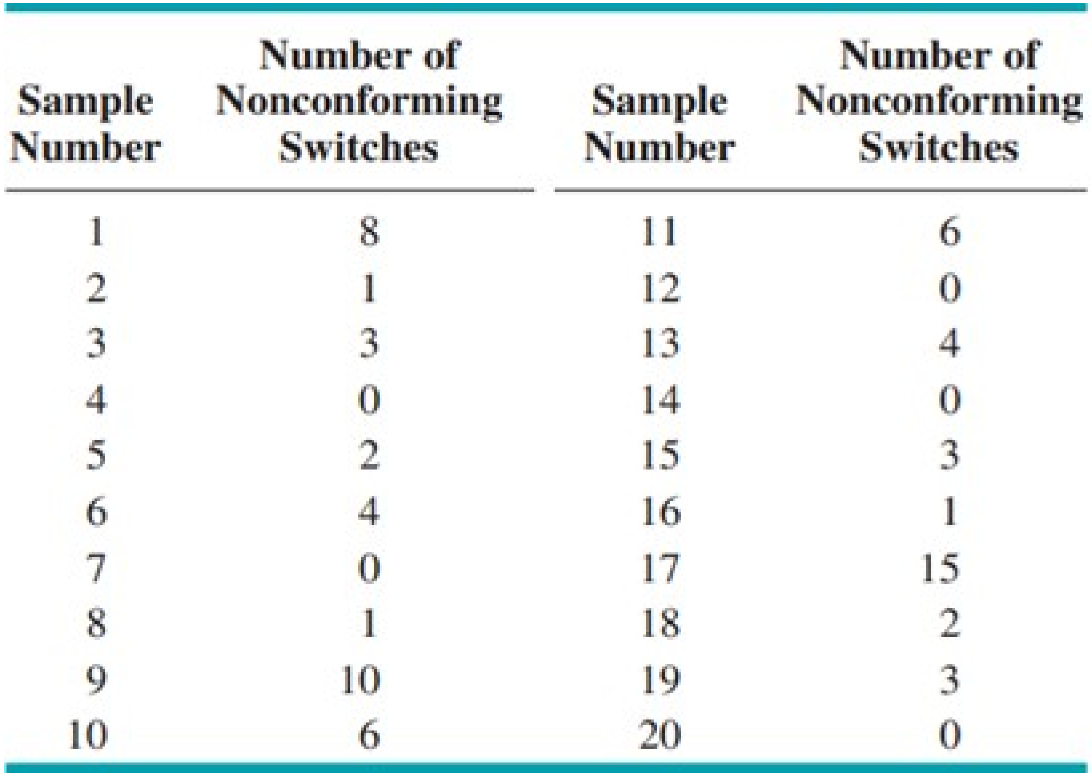
(b) All points on both control charts fall between the control limits computed in part (a). What are the natural tolerance limits of the process?

(c) If the specification limits are 41 ± 5.0, what are your conclusions regarding the ability of the process to produce to produce items within these specifications?

(d) Assuming that if an item exceeds the upper specification limit it can be reworked and if it is below the lower specification limit it must be scrapped, what percent scrap and rework is the process producing?

The number of nonconforming switches in samples of size 150 are shown in the following table. Construct a fraction nonconforming control chart for these data. Does the process appear to be in control?

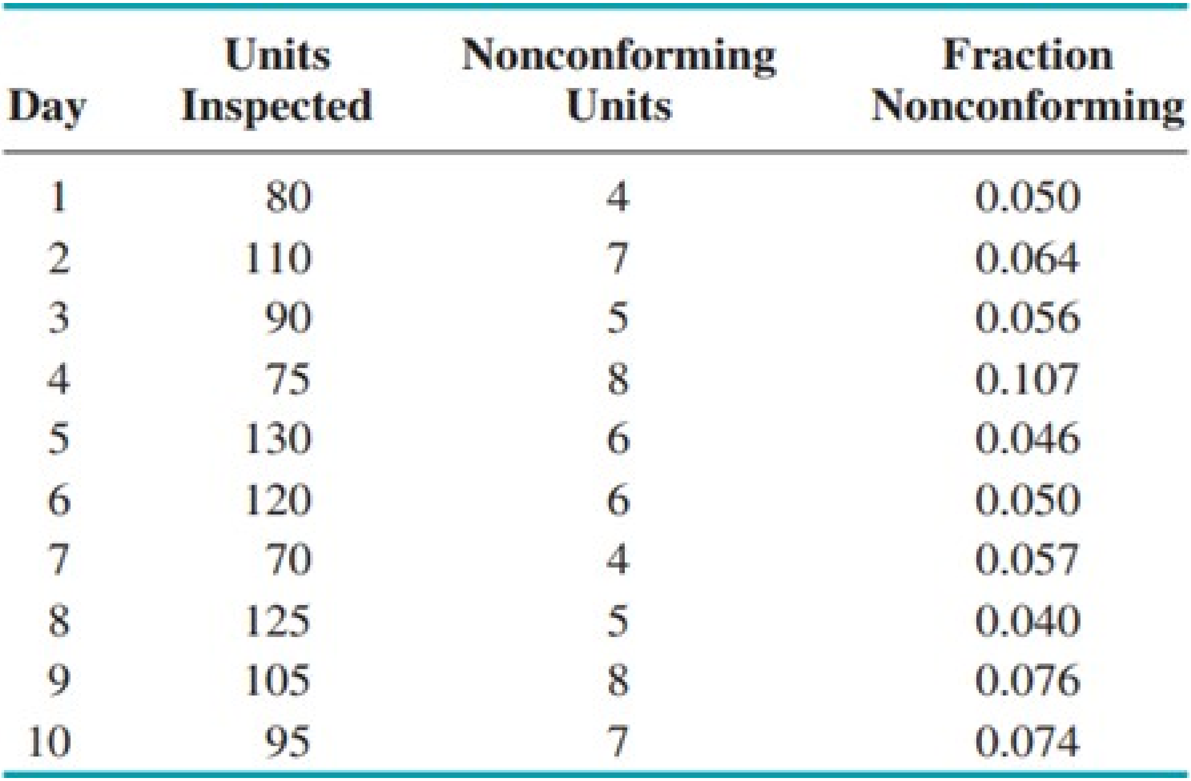
Number of Nonconforming Switches



Sample Number	Number of Nonconforming Switches	Sample Number	Number of Nonconforming Switches
1	8	11	6
2	1	12	0
3	3	13	4
4	0	14	0
5	2	15	3
6	4	16	1
7	0	17	15
8	1	18	2
9	10	19	3
10	6	20	0

The Table below represents the results of inspecting all units of a personal computer produced for the past ten days. Does the process appear to be in control?

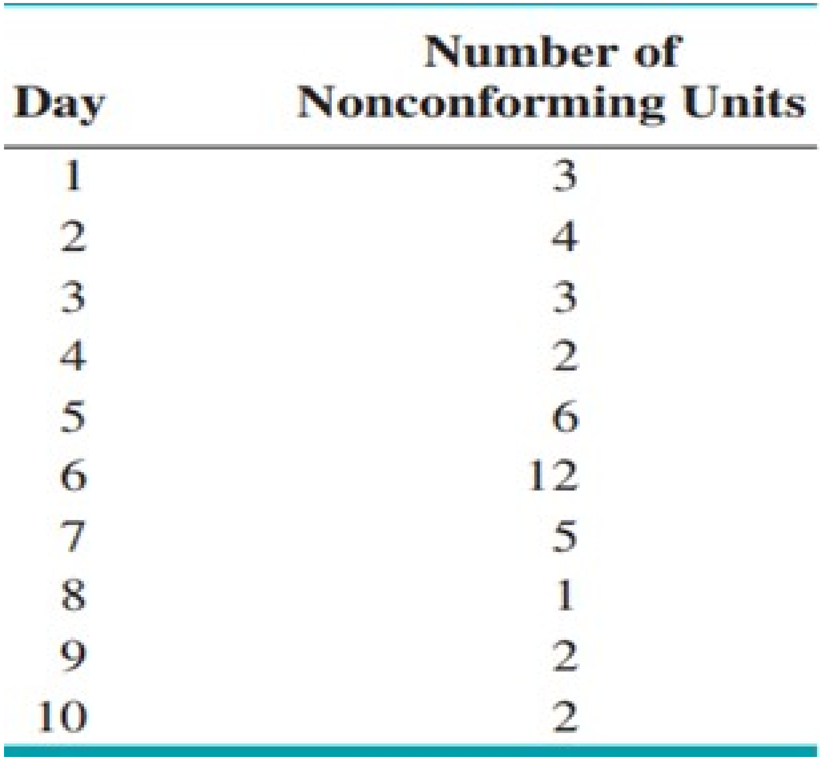
Personal Computer Inspecting Results



Day	Units Inspected	Nonconforming Units	Fraction Nonconforming
1	80	4	0.050
2	110	7	0.064
3	90	5	0.056
4	75	8	0.107
5	130	6	0.046
6	120	6	0.050
7	70	4	0.057
8	125	5	0.040
9	105	8	0.076
10	95	7	0.074

Based on the data in the given Table if an np chart is to be established, what would you recommend as the center line and control limits?Assume that n = 500.

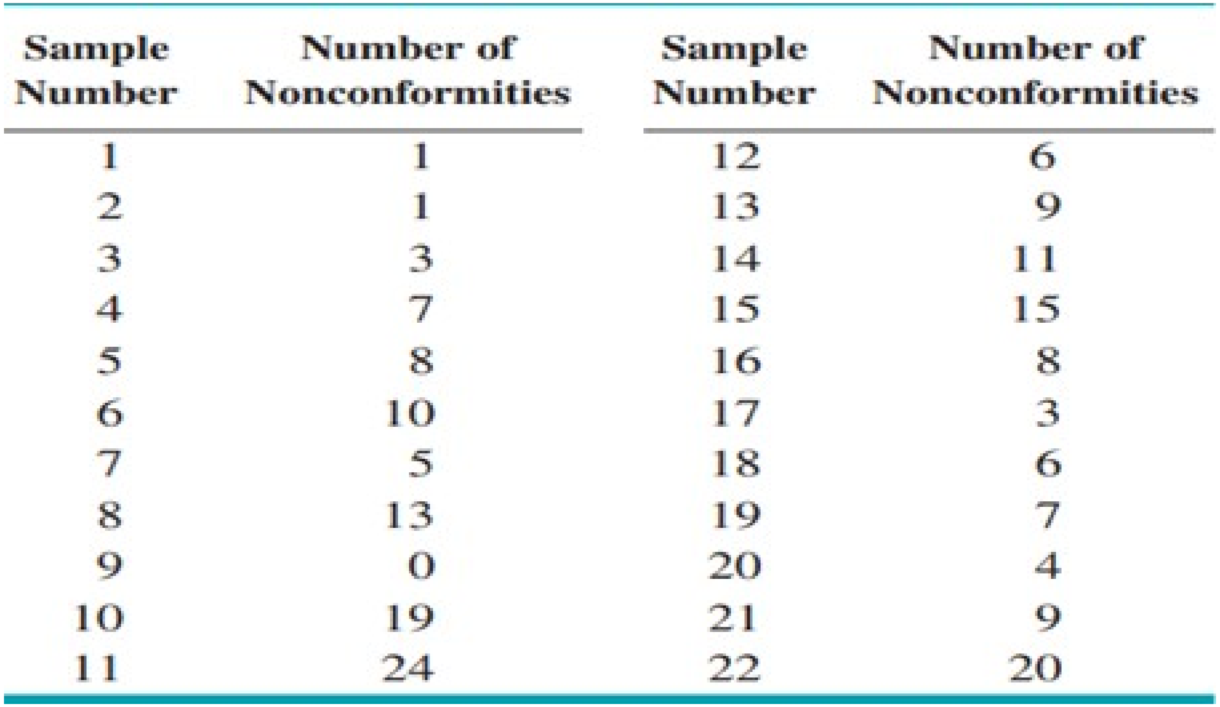
Data for the Exercise



Day	Number of Nonconforming Units
1	3
2	4
3	3
4	2
5	6
6	12
7	5
8	1
9	2
10	2

The tabulated data represent the number of nonconformities per 1000 meters in telephone cable. From analysis of these data would you conclude that the process is in statistical control? What control procedure would you recommend for future production?

Telephone Cable Data for the exercise



Sample Number	Number of Nonconformities	Sample Number	Number of Nonconformities
1	1	12	6
2	1	13	9
3	3	14	11
4	7	15	15
5	8	16	8
6	10	17	3
7	5	18	6
8	13	19	7
9	0	20	4
10	19	21	9
11	24	22	20

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

BSDCHZC355
Statistical Inference & Applications

Shaibal Kr. Sen
Session 16 (Contact Hr 1& 2)

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

STATISTICAL INFERENCES

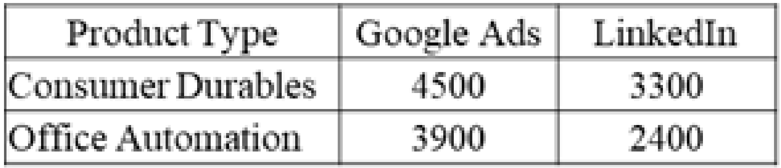
&

APPLICATIONS

(Session 16)

➢REVISION SESSION

Ex 1: A leading Marketing Consultancy Group conducted a research to investigate how product type and advertisement platform affect the success of their campaigns. An online advertisement campaign was undertaken for two types of products viz. - Consumer Durables and Office Automation, on two different platforms - Google Ads and LinkedIn Ads and recorded the average number of viewers for each combination as tabulated below.

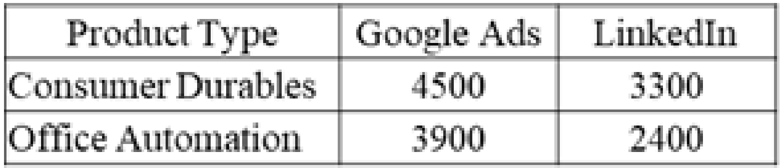


Product Type	Google Ads	LinkedIn
Consumer Durables	4500	3300
Office Automation	3900	2400

The Consultancy Group want to analyze - whether the platform (Google Ads vs. LinkedIn Ads) matters, whether product type matters and whether there is an interaction (i.e., if effect of platform depends on the product type).

(a) Based on the given averages, is there any visible interaction between product type and platform? Explain briefly.

(b) Which factor appears to have a stronger influence on the number of views: the platform used or the product type? Support your answer with reasoning



Product Type	Google Ads	LinkedIn
Consumer Durables	4500	3300
Office Automation	3900	2400

Soln : (a) It is evident that the performance drops from Google Ads to LinkedIn for each product as below -

Consumer Durables : 4500 – 3300 = 1200

Office Automation : 3900 – 2400 = 1500

Since the drop in views is different for the two products, this suggests that the platform's effect depends on the product type. Therefore, it can be inferred that there appears to be an interaction between platform and product type.

(b) We can compare magnitude of changes to verify which factor has stronger influence -

Difference between products on Google Ads = 4500 − 3900 = 600

Difference between products on LinkedIn = 3300 − 2400 = 900

But across platforms, the drop is larger -

Consumer Durables : 1200

Office Automation : 1500

This shows that the platform, Google Ads and LinkedIn causes a bigger change in reach than product type which means platform has a stronger effect on viewership.

Ex 2 **:** In a Completely Randomized Design (CRD), a laboratory tested the effectiveness of three cleaning solutions - CS_1, CS_2 and CS_3 on stain removal. Each solution was randomly applied to four glass slides and the stain removal effectiveness was rated on a scale from 1 to 10. The results were as follows -

Solution CS_1 : 8, 9, 8, 10 = 35

Solution CS_2 :4, 5, 6, 7 = 22

Solution CS_3 : 6, 7, 8, 8 = 29

(a) Compute the Correction Factor

(b) Compute the Total Sum of Squares (TSS)

Soln : (a) Grand Total (G) = 8 + 9 + 8 + 10 + 4 + 5 + 6 + 7 + 6 + 7 + 8 + 8 = 86

$CF = G^2 / N = 86^2/12 \approx 616.33$ (Here, G = 86 and N = 12)

(b) $TSS = \Sigma(Y_{ij})^2 - CF$

$= (8^2 + 9^2 + 8^2 + 10^2 + 4^2 + 5^2 + 6^2 + 7^2 + 6^2 + 7^2 + 8^2 + 8^2) - 616.33$

$= (64 + 81 + 64 + 100 + 16 + 25 + 36 + 49 + 36 + 49 + 64 + 64) - 616.33$

$= 648 - 616.33 = 31.67$

Ex 3 **:** An experimenter performs a Randomized Block Design where six different seeding rates (Treatments T_1 to T_6) were tested over four field blocks. The following summary values were provided from the analysis:

Grand Total (G) = 285.36

Correction Factor (CF) = 1413.72

Total Sum of Squares (TSS) = 12.048

Sum of Squares for Blocks (SSB) = 4.716

Sum of Squares for Treatments (SST) = 3.042

(a) Calculate the Error Sum of Squares (SSE).

(b) Based on the given sums of squares, which factor - blocks or treatments has a larger impact on variation? Justify your answer.

Soln **:** (a) SSE = TSS − SSB − SST

= 12.048 − 4.716 − 3.042 = 4.29

(b) Since SSB = 4.716 is larger than SST = 3.042, we conclude that blocks have a stronger effect on the response variable than the treatments

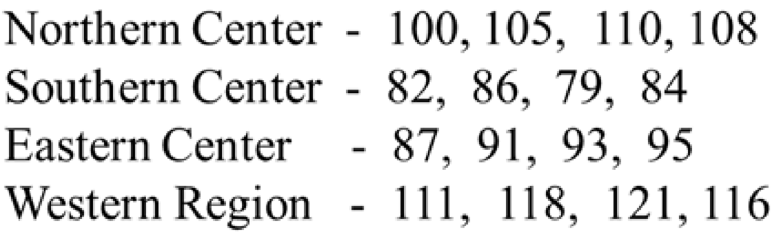
Ex 4 : Monthly sales (in lakhs of Rs.) of a distributor in the four Regional Centers during consecutive four months are given below –

Northern Center	-	100, 105, 110, 108
Southern Center	-	82, 86, 79, 84
Eastern Center	-	87, 91, 93, 95
Western Region	-	111, 118, 121, 116

Using one-way ANOVA, analyse if there is any noticeable difference in sales among the Regions.

(a) Compute the Grand Mean of Sales based on the given sales figures for the month.

(b) Based only on the group averages, is there likelihood of significant difference between the sales of the regions? Justify your answer.


Northern Center - 100, 105, 110, 108
Southern Center - 82, 86, 79, 84
Eastern Center - 87, 91, 93, 95
Western Region - 111, 118, 121, 116

Soln : Grand Total of sales of all the four regions
= 100 + 105 + 110 + 108 + 82 + 86 + 79 + 84 + 87 + 91 + 93 + 95 + 111 + 118 + 121 + 116 = 1586

Number of observations = 16

Hence, Grand Mean = Total Sales / Number of observations
= 1586/16 ≈ 99.13

(b) Group mean sales of the different Regional Centers is :
Northern Center - (100 + 105 + 110 + 108)/4 = 423/4 = 105.75
Southern Center - (82 + 86 + 79 + 84)/4 = 331/4 = 82.75
Eastern Center - (87 + 91 + 93 + 95)/4 = 366/4 = 91.25
Western Region - (111 + 118 + 121 + 116)/4 = 466/4 = 116.5

The difference between group means is quite large. West average (Rs. 116.5 Lakhs) is much higher & South (Rs. 82.75 L) is much lower than the grand mean (Rs. 99.13 L). Therefore, it is likely that ANOVA would detect a statistically significant difference between these groups.

Ex 5 : An experiment with Completely Randomized Design was conducted to study the effect of three different fertilizers – M_1, M_2 and M_3 on crop yield in 3 homogeneous plots. Each fertilizer was randomly applied to three homogeneous plots. The yields (in kg) from each treatment are given below -

M_1 **:** 7.75, 8.25, 8

M_2 **:** 9.5, 9.25, 9.75

M_3 **:** 7.25, 7, 7.5

(a)Calculate the grand total of all the observations.

(b) Using treatment totals, compute the sum of squares due to treatment (SST).

M_1 : 7.75, 8.25, 8

M_2 : 9.5, 9.25, 9.75

M_3 : 7.25, 7, 7.5

Soln : (a) Total yield due to M_1 = 7.75 + 8.25 + 8 = 24

Total yield due to M_2 = 9.5 + 9.25 + 9.75 = 28.5

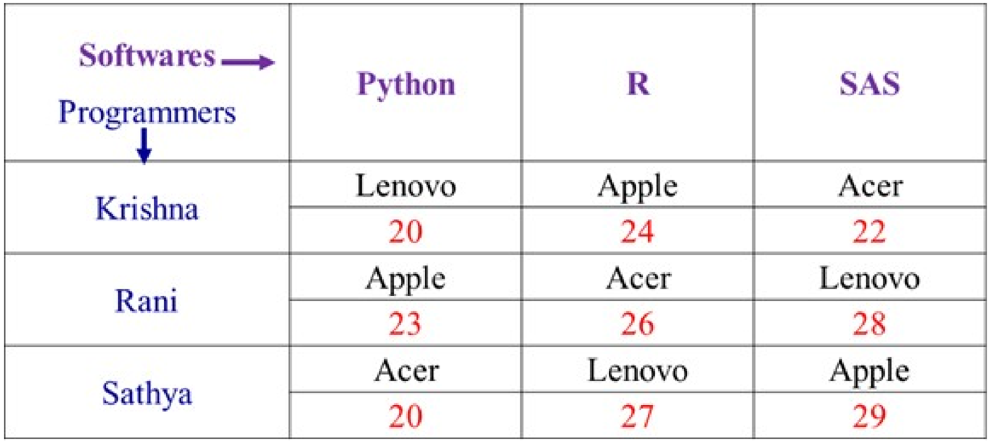
Total yield due to M_3 = 7.25 + 7 + 7.5 = 21.75

Grand Total (G) = 24 + 28.5 + 21.75 = 74.25

(b) $SST = (24^2 + 28.5^2 + 21.75^2)/3 - G^2/3^2$

$\approx 620.44 - 74.25^2/9 = 620.44 - 612.56 = 7.88$

Ex 6 : In LSD, blocking (local control principle) is applied in two perpendicular directions. The output times of a program for a common problem written by 3 different Programmers - Krishna, Rani, Sathya (Rows) using 3 different Softwares - Python, R, SAS (Columns) executed in three different Brands of Laptops - Apple, Lenovo, Acer are given in the following design.



Softwares → Programmers ↓	Python	R	SAS
Krishna	Lenovo	Apple	Acer
	20	24	22
Rani	Apple	Acer	Lenovo
	23	26	28
Sathya	Acer	Lenovo	Apple
	20	27	29

Note :
Softwares in violet
Programmers in Blue
Computer Brands in Black
Output time in Red

(a) Compute the grand total and verify the total number of observations (N).
(b) Based on the software-wise totals, which software appears to have the lowest average execution time? Show your reasoning.

Softwares → Programmers ↓	Python	R	SAS
Krishna	Lenovo 20	Apple 24	Acer 22
Rani	Apple 23	Acer 26	Lenovo 28
Sathya	Acer 20	Lenovo 27	Apple 29

Soln **:** (a) Total = 20 + 24 + 22 + 23 + 26 + 28 + 20 + 27 + 29 = 219

Total Number of observations (N) = 3×3 = 9

(b) Software wise Times Average Times

Python = 20 + 23 + 20 = 63 ⇒ 63/3 = 21.0

R = 24 + 26 + 27 = 77 ⇒ 77/3 ≈ 25.67

SAS = 22 + 28 + 29 = 79 ⇒ 79/3 ≈ 26.33

Hence, Python takes the lowest average execution time ie. it is the fastest.

Ex 7 **:** Perform ANOVA of the following LSD.

An experiment was carried out to determine the effect of claying the ground on the field of barley grains; amount of clay used were as follows –

A : No clay.

B : Clay at 100 per acre.

C : Clay at 200 per acre.

D : Clay at 300 per acre.

The yields were in plots of 8 meters by 8 meters and are given in the table.

D **29.1**	B **18.9**	C **29.4**	A **5.7**
C **16.4**	A **10.2**	D **21.2**	B **19.1**
A **5.4**	D **38.8**	B **24.0**	C **37.0**
B **24.9**	C **41.7**	A **9.5**	D **28.9**

Note : Figures in Bold Blue shows the yield of the plots.

A, B, C & D are plots of grounds.

Soln :

H_{0r} : Row means are all equal - vs- H_{1r} : Row means are not all equal.

H_{0c} : Column means are all equal - vs- H_{1c} : Column means are not all equal.

H_{0t} : Treatment means are all equal - vs- H_{1t} : Treatment means are not all equal.

	I	II	III	IV	Row Totals (R_i)
I	D 29.1	B 18.9	C 29.4	A 5.7	83.1
II	C 16.4	A 10.2	D 21.2	B 19.1	66.9
III	A 5.4	D 38.8	B 24	C 37	105.2
IV	B 24.9	C 41.7	A 9.5	D 28.9	105
Column Totals (C_j)	75.8	109.6	84.1	90.7	360.2 = G

	I	II	III	IV	Row Totals (Ri)
I	D 29.1	B 18.9	C 29.4	A 5.7	83.1
II	C 16.4	A 10.2	D 21.2	B 19.1	66.9
III	A 5.4	D 38.8	B 24	C 37	105.2
IV	B 24.9	C 41.7	A 9.5	D 28.9	105
Column Totals (Cj)	75.8	109.6	84.1	90.7	360.2 = G

The four treatment totals are –

A : 30.8

(5.7 + 10.2 + 5.4 + 9.5 = 30.8)

B : 86.9

(18.9 + 19.1 + 24 + 24.9 = 86.9)

C : 124.5

(29.4 + 16.4 + 37 + 41.7 = 124.5)

D : 118.0

(29.1 + 21.2 + 38.8 + 28.9 = 118)

Grand Total G = 360.2, $N = m^2 = 16$; $CF = G^2/N = 360.2^2/16 = 8109.0025$

Raw SS (RSS) = $\Sigma(y_{ijk})^2 = 29.1^2 + 18.9^2 + \ldots.. + 9.5^2 + 28.9^2 = 10{,}052.08$

Total SS (TSS) = RSS – CF = 10,052.08 – 8109.0025 = 1943.0775

$SSR = \Sigma(R_i)^2/m - CF = (83.1^2 + 66.9^2 + 105.2^2 + 105^2)/4 - 8109.0025 = 259.3125$

$SSC = \Sigma(C_j)^2/m - CF = (75.8^2 + 109.6^2 + 84.1^2 + 90.7^2)/4 - 8109.0025 = 155.2725$

$SST = \Sigma(T_k)^2/m - CF = (30.8^2 + 86.9^2 + 124.5^2 + 118^2)/4 - 8109.0025 = 1372.1225$

Error SS = TSS – SSR – SSC – SST

= 1943.0775 – 259.3125 – 155.2725 – 1372.1225 = 156.37

Grand Total G = 360.2, $N = m^2 = 16$; $CF = G^2/N = 360.2^2/16 = 8109.0025$

Raw SS (RSS) = $\Sigma(y_{ijk})^2 = 29.1^2 + 18.9^2 + \ldots\ldots + 9.5^2 + 28.9^2 = 10{,}052.08$

Total SS (TSS) = RSS – CF = 10,052.08 – 8109.0025 = 1943.0775

$SSR = \Sigma(R_i)^2/m - CF = (83.1^2 + 66.9^2 + 105.2^2 + 105^2)/4 - 8109.0025 = 259.3125$

$SSC = \Sigma(C_j)^2/m - CF = (75.8^2 + 109.6^2 + 84.1^2 + 90.7^2)/4 - 8109.0025 = 155.2725$

$SST = \Sigma(T_k)^2/m - CF = (30.8^2 + 86.9^2 + 124.5^2 + 118^2)/4 - 8109.0025 = 1372.1225$

Error SS = TSS – SSR – SSC – SST

= 1943.0775 – 259.3125 – 155.2725 – 1372.1225 = 156.37

ANOVA Table for LSD

Source of Variation	DF	SS	MSS = SS/DF	F_{cal} = MSS/MSE
Rows	m - 1 = 3	259.3125	86.4375	FR = 3.316
Columns	m - 1 = 3	155.2725	51.7575	FC = 1.986
Treatments	m - 1 = 3	1372.123	457.3742	FT = 17.549
Error	(m - 1)(m - 2) = 6 = 15 - 9	156.37	26.0616	
Total	m^2 - 1 = 15	1943.0775		

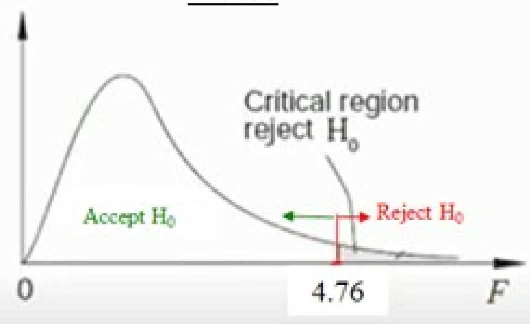
Error DF = $m^2 - 1 - \{(m-1) + (m-1) + (m-1)\}$
$= (m+1)(m-1) - 3(m-1) = (m-1)(m-2)$

$F_{critical}$ value obtained from F-table for $\alpha = 0.05$ and df 3, 6 ie. $F_{0.05,\,3,\,6} = 4.76$. Now, comparing the F_{cal} with $F_{critical}$ we conclude that the null hypothesis for rows and columns can not be rejected whereas for treatments null hypothesis is rejected ie. different levels of clay have significant effect on the yield.

ANOVA Table for LSD

Source of Variation	DF	SS	MSS = SS/DF	F_{cal} = MSS/MSE
Rows	m - 1 = 3	259.3125	86.4375	FR = 3.316
Columns	m - 1 = 3	155.2725	51.7575	FC = 1.986
Treatments	m - 1 = 3	1372.123	457.3742	FT = 17.549
Error	(m - 1)(m - 2) = 6	156.37	26.0616	
Total	m^2 - 1 = 15	1943.0775		

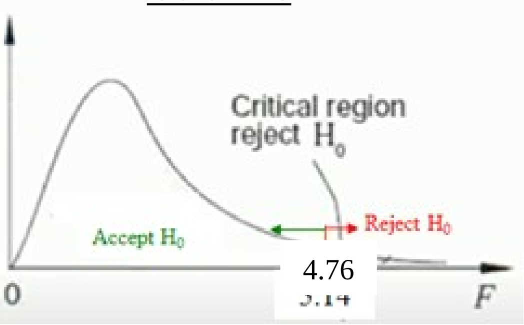
Rows


Critical region
reject H_0
Accept H_0
Reject H_0
0
4.76
F

F_{cal} (3.316) < $F_{critical}$ (4.76)

So, Null hypothesis can not be rejected.

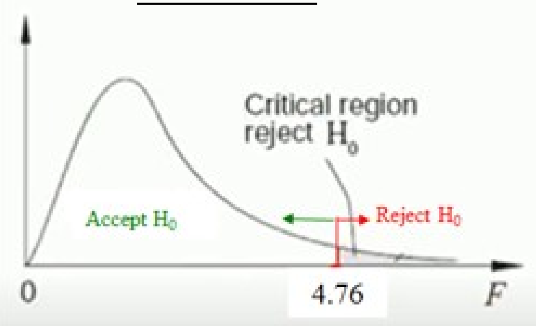
Columns


Critical region
reject H_0
Accept H_0
Reject H_0
0
4.76
F

F_{cal} (1.986) < $F_{critical}$ (4.76)

So, Null hypothesis can not be rejected.

Treatments


Critical region
reject H_0
Accept H_0
Reject H_0
0
4.76
F

F_{cal} (17.549) > $F_{critical}$ (4.76)

So, Null hypothesis can not be accepted.

Ex 8 : Analyze the following BIBD

Balanced Incomplete Block Design for Catalyst Experiment

Treatment (Catalyst)	Block (Batch of Raw Material)			
	1	2	3	4
1	73	74	-	71
2	-	75	67	72
3	73	75	68	-
4	75	-	72	75

B1	B2	B3	B4
1	1	2	1
3	2	3	2
4	3	4	4

H_{0t} : All treatment effects are equal -vs- H_{at} : All treatment effects are not equal

H_{0b} : All block effects are equal -vs- H_{ab} : All block effects are not equal

Soln :

Treatment (v) = 4, Block (b) = 4, Each treatment occurs in blocks (r) = 3, Block size k < v = 3, Each pair of treatment occurring (λ) = 2

Balanced Incomplete Block Design for Catalyst Experiment

Treatment (Catalyst)	Block (Batch of Raw Material)				
	1	2	3	4	$y_{i.}$
1	73	74	-	71	218
2	-	75	67	72	214
3	73	75	68	-	216
4	75	-	72	75	222
$y_{.j}$	221	224	207	218	$870 = y_{..}$

Treatment (Catalyst)	Block (Batch of Raw Material)				
	1	2	3	4	$y_{i.}$
1	73	74	-	71	218
2	-	75	67	72	214
3	73	75	68	-	216
4	75	-	72	75	222
$y_{.j}$	221	224	207	218	$870 = y_{..}$

Treatment (v) = 4, Block (b) = 4, Each treatment occurs in blocks (r) = 3, Block size k < v = 3, Each pair of treatment occurring (λ) = 2

This is a BIBD with $v = a = 4$, $b = 4$, $k = 3$, $r = 3$, $\lambda = 2$ and $N = 12$. The analysis of this data is as follows. The Total Sum of Squares is –

$$SST = \Sigma\,(\text{obsn})^2 - G^2/N = \sum_i \sum_j (y_{ij})^2 - (y_{..})^2/12$$

$$= (73^2 + 74^2 + \ldots + 75^2) - 870^2/12 = 63156 - 63075 = 81$$

$$SSBlocks = 1/3\Big[\sum_{j=1}^{4} (y_{.j})^2\Big] - G^2/N = 1/3[221^2 + 224^2 + 207^2 + 218^2] - 870^2/12$$

$$= 63125 - 63070 = 55$$

$$Q_1 = T_1 - 1/3(B_1 + B_2 + B_4) = 218 - 1/3(221 + 224 + 218) = -\,9/3 = -\,3$$

$$Q_2 = T_2 - 1/3(B_2 + B_3 + B_4) = 214 - 1/3(224 + 207 + 218) = -\,7/3$$

$$Q_3 = T_3 - 1/3(B_1 + B_2 + B_3) = 216 - 1/3(221 + 224 + 207) = -\,4/3$$

$$Q_4 = T_4 - 1/3(B_1 + B_3 + B_4) = 222 - 1/3(221 + 207 + 218) = 20/3$$

This is a BIBD with $v = a = 4$, $b = 4$, $k = 3$, $r = 3$, $\lambda = 2$ and $N = 12$. The analysis of this data is as follows. The Total Sum of Squares is –

$$SST = \Sigma\,(\text{obsn})^2 - G^2/N = \sum_i \sum_j (y_{ij})^2 - (y_{..})^2/12$$

$$= (73^2 + 74^2 + \ldots + 75^2) - 870^2/12 = 63156 - 63075 = 81$$

$$SSBlocks = 1/3[\sum_{j=1}^{4} (y_{.j})^2] - G^2/N = 1/3[221^2 + 224^2 + 207^2 + 218^2] - 870^2/12$$

$$= 63125 - 63070 = 55$$

$Q_1 = T_1 - 1/3(B_1 + B_2 + B_4) = 218 - 1/3(221 + 224 + 218) = - 9/3 = - 3$

$Q_2 = T_2 - 1/3(B_2 + B_3 + B_4) = 214 - 1/3(224 + 207 + 218) = - 7/3$

$Q_3 = T_3 - 1/3(B_1 + B_2 + B_3) = 216 - 1/3(221 + 224 + 207) = - 4/3$

$Q_4 = T_4 - 1/3(B_1 + B_3 + B_4) = 222 - 1/3(221 + 207 + 218) = 20/3$

Balanced Incomplete Block Design for Catalyst Experiment

Treatment (Catalyst)	Block (Batch of Raw Material)				
	1	2	3	4	$y_{i.}$
1	73	74	-	71	218
2	-	75	67	72	214
3	73	75	68	-	216
4	75	-	72	75	222
$y_{.j}$	221	224	207	218	$870 = y_{..}$

$$\text{SSTreatment (adjusted)} = \text{SSTreat.}_{(adj)} = [k\sum_{i=1}^{4} (Q_i)^2]/\lambda a = \Sigma\, \hat{\tau}_i\, Q_i$$

$$= 3[9-9/3)2 + (-7/3)2 + (-4/3)2 + (20/3)2]/(2x4) = 22.75$$

The error sum of squares is obtained by subtraction as -

$$SSE = SST - \text{SSTreat}_{.(adj)} - SSBlocks = 81 - 22.75 - 55 = 3.25$$

$Q'_1 = B_1 - 1/3(T_1 + T_3 + T_4) = 221 - 1/3(218 + 216 + 222) = 7/3$

$Q'_2 = B_2 - 1/3(T_1 + T_2 + T_3) = 224 - 1/3(218 + 214 + 216) = 24/3$

$Q'_3 = B_3 - 1/3(T_2 + T_3 + T_4) = 207 - 1/3(214 + 216 + 222) = -31/3$

$Q'_4 = B_4 - 1/3(T_1 + T_2 + T_4) = 218 - 1/3(218 + 214 + 222) = 0$

$SSBlocks_{(adj.)} = [r\Sigma (Q'_i)^2]/\lambda b = 3[(7/3)^2 + (24/3)^2 + (-31/3)^2 + (0)^2]/(2x4) = 66.08$

$SSTreat. = \Sigma (T_i)^2/r – G^2/N = [(218)^2 + (214)^2 + (216)^2 + (222)^2]/3 – (870)^2/12$

$= 11.67$

ANOVA including both Treatments & Blocks

Source of Variation	DF	SS	MS	$F_{calculated}$	$F_{critical}$
SSTreatments (adj.)	3	22.75	22.75/3 = 7.58	7.58/0.65 = 11.66	5.41
SSTreatments (unadj.)	3	11.67			
SSBlocks (unadj.)	3	55.00			
SSBlocks (adj.)	3	66.08	66.08/3 = 22.03	22.03/0.65 = 33.89	5.41
SSError	11 - 6 = 5	3.25	3.25/5 = 0.65		
SSTotal	11	81.00			

Value of $F_{3,5}$ for 5% LOS, obtained from F-table, is 5.41 ie. $F_{critical} = 5.41$

Since both Treatments & Blocks $F_{calculated}$ (11.66) & (33.89) > $F_{critical}$ (5.41), Reject both H_{0t} and H_{0b}

Ex 9 **:** The following table gives the synthetic yields per plot of an experiment conducted with $3^2 = 9$ treatments using a simple lattice (2 replication) design.

	Replication 1				Replication 2		
Blocks	Treatments (yield per plot)			Blocks	Treatments (yield per plot)		
1	1 (8)	7 (5)	4 (3)	4	8 (2)	7 (2)	9 (7)
2	3 (3)	6 (2)	9 (6)	5	4 (3)	5 (3)	6 (3)
3	8 (3)	5 (7)	2 (3)	6	2 (2)	3 (4)	1 (6)

Analyze the data. LOS 5%

Soln :

Grand Total, G = 8 + 5 + … + 6 = 72

No. of observations, n $= 2s^2 = 2 \times 3^2 = 18$

Grand Mean, $\bar{y} = G/n = 72/18 = 4$, No. of replications = 2 (as it is a simple lattice)

Block size, k = 3, Correction Factor (CF) = $G^2/n = 72^2/18 = 288$

	Replication 1				Replication 2		
Blocks	Treatments (yield per plot)			Blocks	Treatments (yield per plot)		
1	1 (8)	7 (5)	4 (3)	4	8 (2)	7 (2)	9 (7)
2	3 (3)	6 (2)	9 (6)	5	4 (3)	5 (3)	6 (3)
3	8 (3)	5 (7)	2 (3)	6	2 (2)	3 (4)	1 (6)

Note : $T_1 = 8 + 6 = 14$, etc.

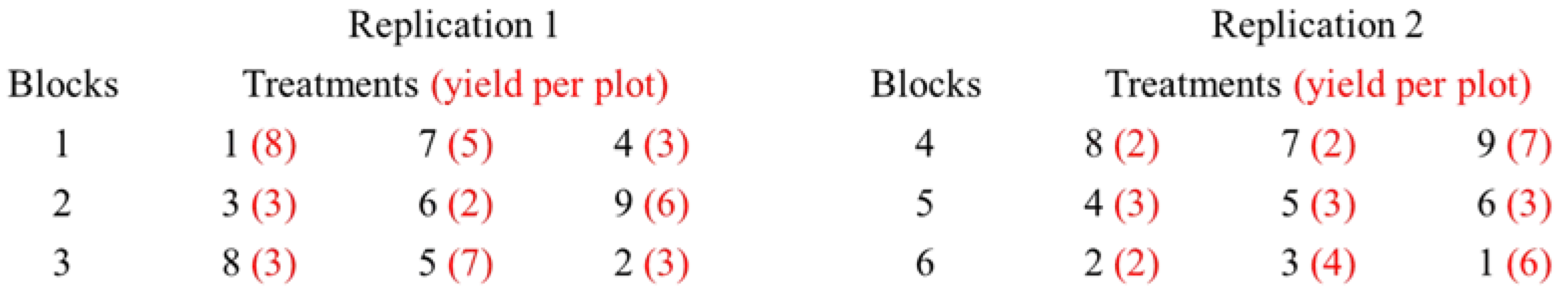
$B_1 = 8 + 5 + 3 = 16$

Total of Block, in which treatment 1 occurs,

$$\sum_{j(1)} B_j = (8 + 5 + 3) + (2 + 4 + 6) = 16 + 12 = 28, \text{ etc.}$$

Adjusted Treatment Total, $Q_1 = T_1 - \sum_{j(1)} B_j/k = 14 - 28/3 = 4.67$

$$\hat{\tau}_1 = Q_1/2 + [S_R(Q_1) + S_C(Q_1)]/(2x3) = 4.67/2 + 1.001/6 = 2.50$$



	Replication 1				Replication 2		
Blocks	Treatments (yield per plot)			Blocks	Treatments (yield per plot)		
1	1 (8)	7 (5)	4 (3)	4	8 (2)	7 (2)	9 (7)
2	3 (3)	6 (2)	9 (6)	5	4 (3)	5 (3)	6 (3)
3	8 (3)	5 (7)	2 (3)	6	2 (2)	3 (4)	1 (6)

$T_2 = 3 + 2 = 5$, $T_6 = 2 + 3 = 5$, $T_9 = 6 + 7 = 13$

$B_2 = 3 + 2 + 6 = 11$, $B_4 = 2 + 2 + 7 = 11$, $B_6 = 2 + 4 + 6 = 12$

$\sum_{j=3} B_j = 3 + 2 + 6 + 2 + 4 + 6 = 23$; $\sum_{j=6} B_j = 3 + 2 + 6 + 3 + 3 + 3 = 20$

Replication 1				Replication 2			
Blocks	Treatments (yield per plot)			Blocks	Treatments (yield per plot)		
1	1 (8)	7 (5)	4 (3)	4	8 (2)	7 (2)	9 (7)
2	3 (3)	6 (2)	9 (6)	5	4 (3)	5 (3)	6 (3)
3	8 (3)	5 (7)	2 (3)	6	2 (2)	3 (4)	1 (6)

Specimen examples : $T_1 = 8 + 6 = 14, \ldots, T_5 = 7 + 3 = 10, \ldots, T_9 = 6 + 7 = 13$

$B_1 = 8 + 5 + 3 = 16, \ldots, B_4 = 2 + 2 + 7 = 11, \ldots, B_6 = 2 + 4 + 6 = 12$

$\sum_{j(1)} B_j = (8 + 5 + 3) + (2 + 4 + 6) = 16 + 12 = 28$, (Total of Block, in which treatment 1 occurs)…,

$\sum_{j(5)} B_j = (8 + 5 + 3) + (2 + 4 + 6) = 16 + 12 = 28$ etc.

$\hat{\tau}_1 = Q_1/2 + [S_R(Q_1) + S_C(Q_1)]/(2x3) = 4.67/2 + 1.001/6 = 2.50$

Adjusted treatment mean for treatment i = ith treatment effect ($\hat{\tau}_i$) + Grand Mean $\overline{y}$

Grand Mean, $\overline{y} = G/n = 72/18 = 4$

(1) Treat/ Block No.	(2) T_i	(3) B_j	(4) Block Nos. in which treat i occurs	(5) $\sum_{j(i)} B_j$	(6) $\sum_{j(i)} B_j/k$	(7) Q_i (2) – (6)	(8) $\hat{\tau}_i$	(9) Adj. treat. Mean
1	14	16	1, 6	28	9.33	4.66	2.50	6.50
2	5	11	3, 6	25	8.33	-3.33	-2.16	1.84
3	7	13	2, 6	23	7.66	-0.66	0.33	4.33
4	6	11	1, 5	25	8.33	-2.66	-1.33	2.66
5	10	9	3, 5	22	7.33	2.66	0.50	4.50
6	5	12	2, 5	20	6.66	-1.66	-0.50	3.50
7	7		1, 4	27	9.00	-2.00	-0.83	3.16
8	5		3, 4	24	8.00	-3.00	-2.00	2.00
9	13		2, 4	22	7.33	5.66	3.50	7.50

Replication 1				Replication 2			
Blocks	Treatments (yield per plot)			Blocks	Treatments (yield per plot)		
1	1 (8)	7 (5)	4 (3)	4	8 (2)	7 (2)	9 (7)
2	3 (3)	6 (2)	9 (6)	5	4 (3)	5 (3)	6 (3)
3	8 (3)	5 (7)	2 (3)	6	2 (2)	3 (4)	1 (6)

Total Sum of Squares (TSS) $= \Sigma\,(\text{observation})^2 - CF$

$= 8^2 + 5^2 + \ldots. + 6^2 - 72^2/18 = 66$

Treatment Sum of Squares unadjusted (SST_U) $= [\Sigma\,(T_i)^2]/m - CF$

$= (14^2 + \ldots +13^2)/2 - 72^2/18 = 49$

Block Sum of Squares unadjusted (SSB_U) $= [\Sigma\,(B_j)^2]/s - CF$

$= (16^2 + \ldots + 12^2)/3 - 72^2/18 = 9.33$

Treatment Sum of Squares adjusted (SST_A) $= \Sigma\,\hat{\tau}_i Q_i = 51.44$

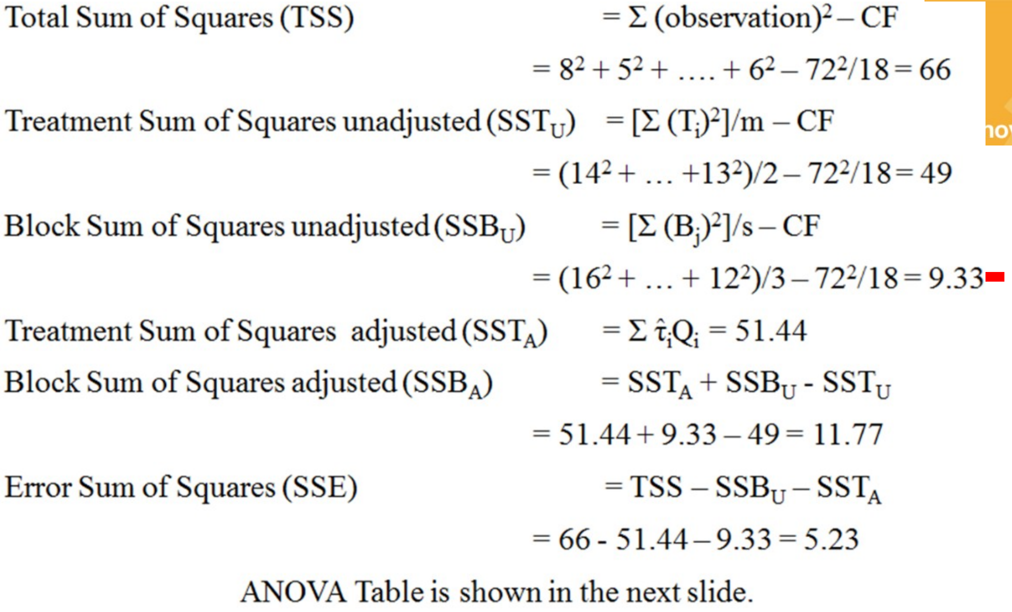
Block Sum of Squares adjusted (SSB_A) $= SST_A + SSB_U - SST_U$

$= 51.44 + 9.33 - 49 = 11.77$

Error Sum of Squares (SSE) $= TSS - SSB_U - SST_A$

$= 66 - 51.44 - 9.33 = 5.23$

ANOVA Table is shown in the next slide.



Total Sum of Squares (TSS) $= \Sigma\,(\text{observation})^2 - CF$

$= 8^2 + 5^2 + \ldots + 6^2 - 72^2/18 = 66$

Treatment Sum of Squares unadjusted (SST_U) $= [\Sigma\,(T_i)^2]/m - CF$

$= (14^2 + \ldots + 13^2)/2 - 72^2/18 = 49$

Block Sum of Squares unadjusted (SSB_U) $= [\Sigma\,(B_j)^2]/s - CF$

$= (16^2 + \ldots + 12^2)/3 - 72^2/18 = 9.33$

Treatment Sum of Squares adjusted (SST_A) $= \Sigma\,\hat{\tau}_i Q_i = 51.44$

Block Sum of Squares adjusted (SSB_A) $= SST_A + SSB_U - SST_U$

$= 51.44 + 9.33 - 49 = 11.77$

Error Sum of Squares (SSE) $= TSS - SSB_U - SST_A$

$= 66 - 51.44 - 9.33 = 5.23$

ANOVA Table is shown in the next slide.

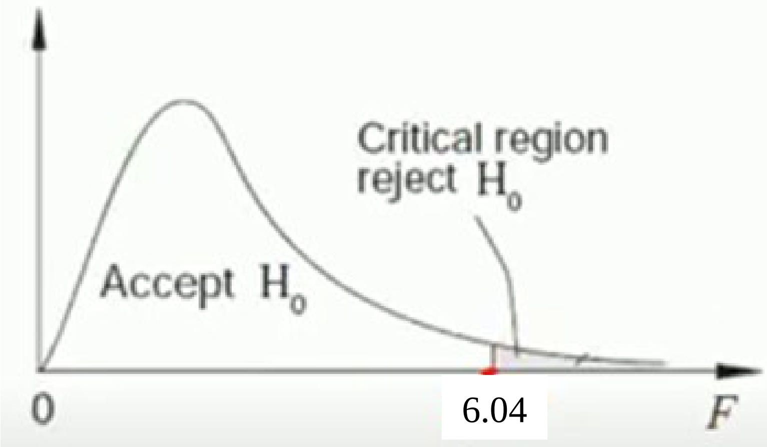
ANOVA Table

Source of Variation	DF	SS	MS	$F_{\text{calculated}}$	F_{critical}
SSBlocks (unadj.)	2x3 – 1 = 5	9.33			
SSTreatments (adj.)	3^2 – 1 = 8	51.44	6.43	4.91	6.04
SSBlocks (adj.)	2x3 – 1 = 5	11.77	2.35	1.79	6.26
SSTreatments (unadj.)	3^2 – 1 = 8	49.00			
SSError	17 - (5 + 8) = 4	5.23	1.31		
SSTotal	2x3^2 – 1 = 17				

$F_{0.05,8,4}$ and $F_{0.05,5,4}$ values obtained from F-table are 6.04 & 6.26 respectively.

As $F_{\text{calculated}}$ values < F_{critical} value we conclude that Treatment effects and block effects are not significantly different.

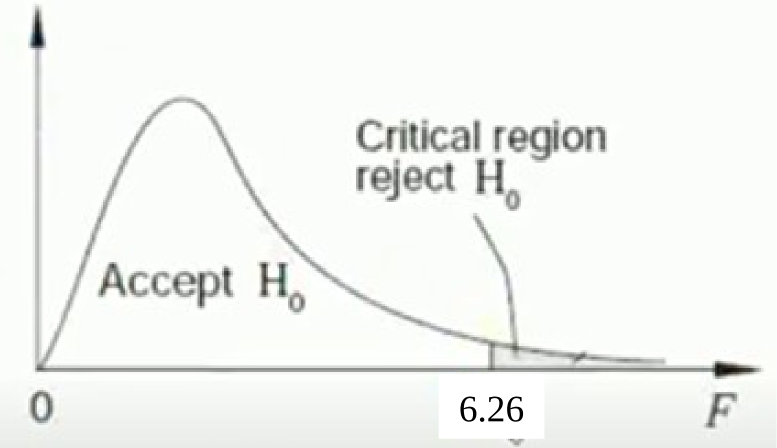
For Treatment



F_{cal} (4.91) < F_{cri} (6.04)

Accept the Null Hypo

For Blocks



F_{cal} (1.79) < F_{cri} (6.26)

Accept the Null Hypo

Ex 10 : A 2^2 experiment in six (Y = 6) randomized blocks was conducted in order to obtain an idea of the interaction : spacing (s) x number of seedlings per hole (N) along with the effects of different types of spacing and different numbers of seedlings per hole, while adopting the Japanese method of cultivation.

The levels of two factors are –

$$S : \begin{cases} 8'' \text{ spacings in between} \\ 10'' \text{ spacings in between} \end{cases}$$

and

$$N : \begin{cases} 3 \text{ seedlings per hole} \\ 4 \text{ seedlings per hole} \end{cases}$$

The field plan and yield of dry Aman paddy (in kg.) are given in next slide.

Block 1

(1)	s	ns	n	Ttl
117	106	109	114	446

Block 2

ns	(1)	s	n	Ttl
114	120	117	114	465

Block 3

(1)	n	s	ns	Ttl
111	117	114	106	448

Block 4

ns	n	s	(1)	
98	121	112	108	439

Block 5

ns	s	(1)	n	
75	97	73	38	283

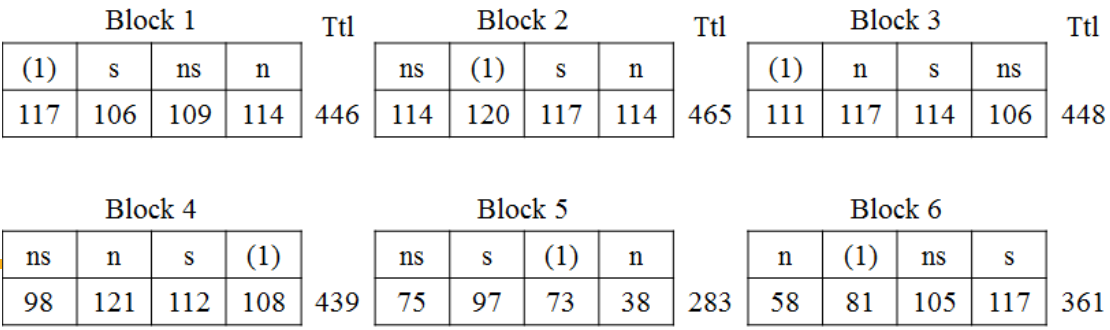
Block 6

n	(1)	ns	s	
58	81	105	117	361

Analyse the data to find out if there are any significant treatment effects – main or interaction.

We apply Yates’ method to find the total effects.

(1)	(1)
a	s
b	n
ab	ns



Block 1				Ttl	Block 2				Ttl	Block 3				Ttl
(1)	s	ns	n		ns	(1)	s	n		(1)	n	s	ns	
117	106	109	114	446	114	120	117	114	465	111	117	114	106	448

Block 4					Block 5					Block 6				
ns	n	s	(1)		ns	s	(1)	n		n	(1)	ns	s	
98	121	112	108	439	75	97	73	38	283	58	81	105	117	361

From slide 19

[S] = – [1]+ [s] – [n] + [sn] = [sn] – [n] + [s] – [1]

[N] = – [1] - [s] + [n] + [sn] = [sn] – [s] + [n] – [1]

[NS] = + [1] - [s] – [n] + [sn] = [sn] – [n] - [s] + [1]

Yates' Method for a 2^2 Experiment

Treatment combination (1)	Total yield from all blocks (2)	(3)	(4)	Average effect
(1)	[1] = 610	[1] + [n] = 1172	[1] + [n] + [s] + [ns] = 2442 = Grand total (G)	
n	[n] = 562	[s] + [ns] = 1270	[n] - [1] + [ns] – [s] = -104 = [N]	[N]/2x6
s	[s] = 663	[n] - [1] = -48	[s] + [ns] - [1] - [n] = 98 = [S]	[S]/2x6
ns	[ns] = 607	[ns] - [s] = -56	[ns] - [s] - [n] + [1] = -8 = [NS]	[NS]/2x6

Now perform the randomized block analysis –

Treatment totals are : [1] = 117 + 120 + 111 + 108 + 73 + 81 = 610

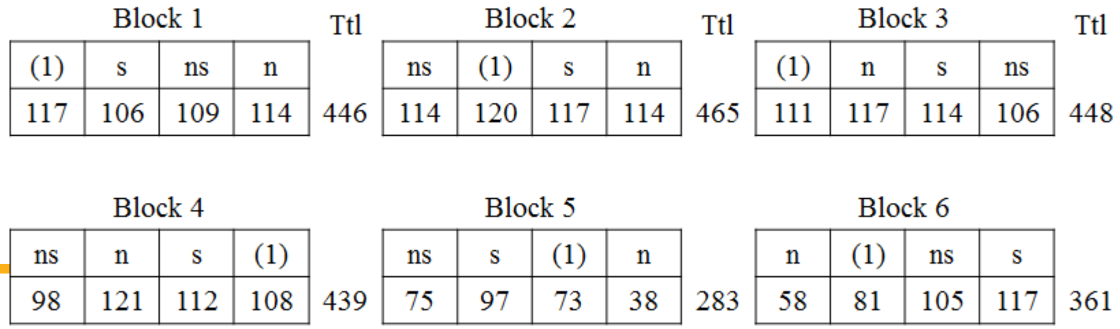
[s] = 106 + 117 + 114 + 112 + 97 + 117 = 663; [n] = 114 + 114 + 117 + 121 + 38 + 58 = 562

and [ns] = 109 + 114 + 106 + 98 + 75 + 105 = 607; G = 610 + 663 + 562 + 607 = 2442

[S] = 607 – 562 + 663 – 610 = 98; [N] = 607 – 663 + 562 – 610 = -104

[NS] = 607 – 562 – 663 + 610 = -8

(1)	(1)
a	s
b	n
ab	ns



Block 1

(1)	s	ns	n	Ttl
117	106	109	114	446

Block 2

ns	(1)	s	n	Ttl
114	120	117	114	465

Block 3

(1)	n	s	ns	Ttl
111	117	114	106	448

Block 4

ns	n	s	(1)	
98	121	112	108	439

Block 5

ns	s	(1)	n	
75	97	73	38	283

Block 6

n	(1)	ns	s	
58	81	105	117	361

Grand Total (G) = [1] + [n] + [s] + [ns] = 610 + 663 + 562 + 607 = 2442

Six block totals are : 446, 465, 448, 439, 283 and 361.

Raw total SS (RSS) = $\Sigma y^2 = 117^2 + 106^2 + \ldots\ldots + 117^2 = 2{,}59{,}024$

Correction Factor (CF) = $G^2/N = (2442)^2/24 = 2{,}48{,}473.5$

Total SS (TSS) = RSS – CF = 2,59,024 – 2,48,473.5 = 10,550.5

Block SS (BSS) = $(446^2 + 465^2 + \ldots + 361^2)/4$ – CF

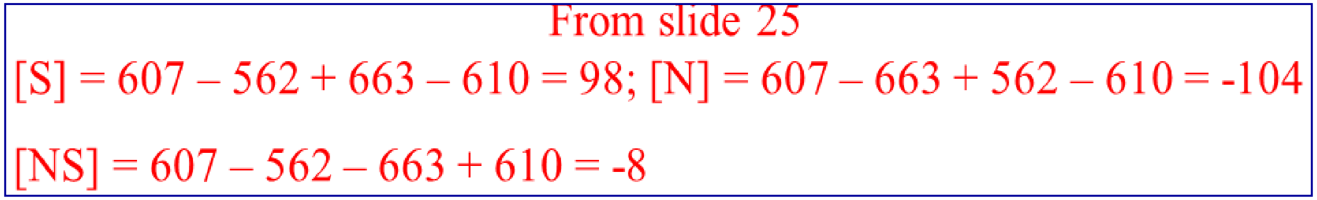
= 2,54,744 – 2,48,473.5 = 6270.5

Treatment SS (TrSS) = $\{(610^2 + \ldots + 607^2)/6\}$ – 2,48,473.5

= 1495962/6 - 2,48,473.5 = 853.5

Error SS = TSS – BSS – TrSS = 10,550.5 – 6,270.5 - 853.5 = 3,426.5

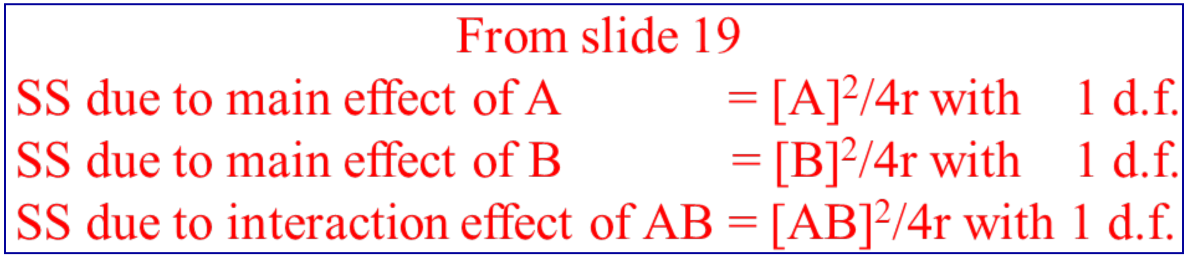
Here, number of Blocks or replicates is 6 and number of treatment combinations is 4. Since, each block contains 4 treatment combinations then each block total square should be divided by 4 while finding block sum of squares. Similarly, each treatment combination replicates as many times as the number of Blocks i.e, 6 then each treatment combination total square should be divided by 6 while finding treatment sum of squares.



From slide 25

$[S] = 607 – 562 + 663 – 610 = 98$; $[N] = 607 – 663 + 562 – 610 = -104$

$[NS] = 607 – 562 – 663 + 610 = -8$



From slide 19

SS due to main effect of A = $[A]^2/4r$ with 1 d.f.

SS due to main effect of B = $[B]^2/4r$ with 1 d.f.

SS due to interaction effect of AB = $[AB]^2/4r$ with 1 d.f.

Also, SS due to N = $[N]^2/4r = (-104)^2/24 = 450.667$

SS due to S = $[S]^2/4r = (98)^2/24 = 400.167$

SS due to NS = $[NS]^2/4r = (-8)^2/24 = 2.67$

(1)	(1)
a	s
b	n
ab	ns

ANOVA Table is presented in next slide

Total SS (TSS) = RSS – CF = 2,59,024 – 2,48,473.5 = 10,550.5

Block SS (BSS) = $(446^2 + 465^2 + \ldots + 361^2)/4$ – CF

= 2,54,744 – 2,48,473.5 = 6270.5

SS due to N = $[N]^2/4r = (-104)^2/24 = 450.667$

SS due to S = $[S]^2/4r = (98)^2/24 = 400.167$

SS due to NS = $[NS]^2/4r = (-8)^2/24 = 2.67$

ANOVA Table for 2^2 experiment

Source of variation	d.f.	SS	MS	$F_{calculated}$	$F_{critical}$
Blocks	5	6,270.50	1254.1		$F_{0.05,1,15}$ = 4.54
N	1	450.667	450.667	1.973	
S	1	400.167	400.167	1.752	
NS	1	2.667	2.667	<1	
Error	23 - 8 = 15	3,426.50	228.433		
Total	6x4 -1 = 23	10,550.50			

There is no significant main or interaction effects present in the above experiment, as in each of the cases the computed value of F is less than the corresponding theoretical value (critical value) at the 5% level of significance.

Ex 11 : A 3^2 experiment was conducted to study the effects of the two factors – Nitrogen (N) & Phosphorus (P) each at 3 levels – 0, 1, 2. Two replications of nine plots each were used. The table shows the Response of the experiment.

Plan and Response of the 3^2 experiment

Replication	Treatment		Response	Replication	Treatment		Response
	n	p			n	p	
I	0	1	14	II	1	2	20
	2	0	15		1	0	19
	0	0	16		1	1	17
	2	1	15		0	0	18
	0	2	16		2	1	19
	1	2	18		0	1	16
	1	1	17		0	2	16
	1	0	19		2	2	19
	2	2	17		2	0	16

(i) Present the Treatment Combination Table
(ii) Treatment Sum of Squares
(iii) Replication Sum of Squares

(i)

Treatment Combinations Table

Replicate	(1)	n	n^2	p	np	n^2p	p^2	np^2	n^2p^2	Total
	00	10	20	01	11	21	02	12	22	
I	16	19	15	14	17	15	16	18	17	147 (R_1)
II	18	19	16	16	17	19	16	20	19	160 (R_2)
Total	34	38	31	30	34	34	32	38	36	307 (G)
	(T_1)	(T_2)	(T_3)	(T_4)	(T_5)	(T_6)	(T_7)	(T_8)	(T_9)	
	[1]	[n]	$[n^2]$	[p]	[np]	$[n^2p]$	$[p^2]$	$[np^2]$	$[n^2p^2]$	

(ii) Treatment Sum of Squares (SST) = $\{\sum(\text{Treatment Total})^2\}/r - CF$

$CF = G^2/n$; Here $r = 2$, $G = 307$ and $n = r(3^2) = 18$

Hence, $CF = G^2/n = 307^2/18 = 5236$

Treatment SS (SST) = $\{\text{Sum (Treatment Total)}^2\}/r - CF$

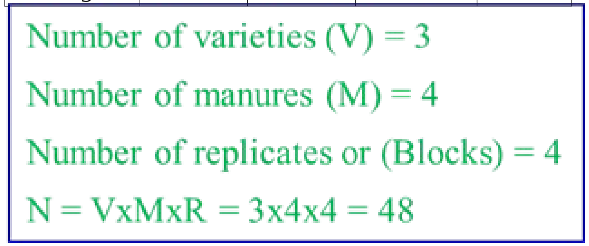
$= (34^2 + 38^2 + \dots + 38^2 + 36^2)/2 - 5236.0556 = 5268.5 - 5236.0556 = 32.4444$

(iii) Replication SS (SSR) = $\{(R_1)^2 + (R_2)^2\}/9 - CF = (147^2 + 160^2)/9 - 5236.0556$

$= 5245.4444 - 5236.0556 = 9.3888$

Ex 12 **:** A variety-manurial experiment was conducted by allotting the three varieties V_1, V_2 and V_3 at random to the plots of four randomized blocks and then, splitting each plot into four sub-plots, the four manures M_1, M_2, M_3 and M_4 were applied at random within each plot. The plan and yield are shown below. Analyse the data to find out if there are any effects due to manure or variety of interaction between variety and manure.

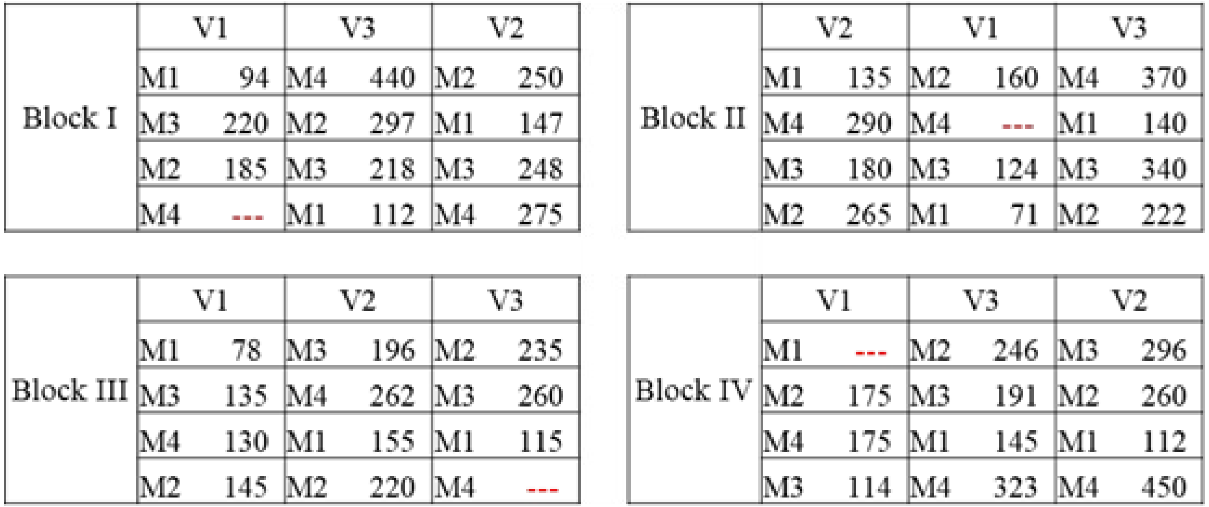
Variety	BLOCK			
	I	II	III	IV
V_1	609	450	488	545
V_2	920	870	833	1118
V_3	1067	1072	1093	905


Number of varieties (V) = 3

Number of manures (M) = 4

Number of replicates or (Blocks) = 4

N = VxMxR = 3x4x4 = 48



	V1		V3		V2	
Block I	M1	94	M4	440	M2	250
	M3	220	M2	297	M1	147
	M2	185	M3	218	M3	248
	M4	---	M1	112	M4	275

	V2		V1		V3	
Block II	M1	135	M2	160	M4	370
	M4	290	M4	---	M1	140
	M3	180	M3	124	M3	340
	M2	265	M1	71	M2	222

	V1		V2		V3	
Block III	M1	78	M3	196	M2	235
	M3	135	M4	262	M3	260
	M4	130	M1	155	M1	115
	M2	145	M2	220	M4	---

	V1		V3		V2	
Block IV	M1	---	M2	246	M3	296
	M2	175	M3	191	M2	260
	M4	175	M1	145	M1	112
	M3	114	M4	323	M4	450

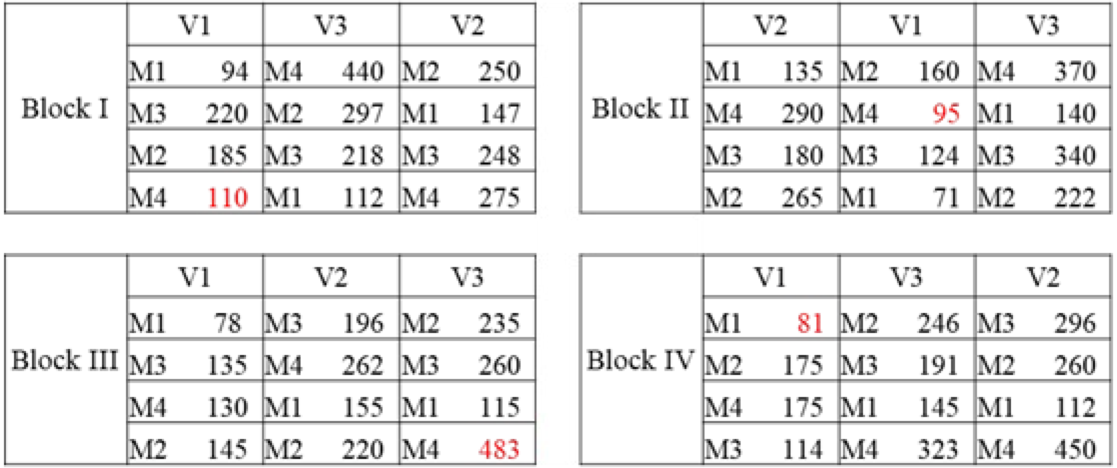
(i) Specify the missing datas (indicated by red dash ---).

(ii) Draw up the block-variety table

(iii) Find the correction factor.

Variety	BLOCK			
	I	II	III	IV
V_1	609	450	488	545
V_2	920	870	833	1118
V_3	1067	1072	1093	905

Soln :



Block I	V1		V3		V2	
	M1	94	M4	440	M2	250
	M3	220	M2	297	M1	147
	M2	185	M3	218	M3	248
	M4	110	M1	112	M4	275

Block II	V2		V1		V3	
	M1	135	M2	160	M4	370
	M4	290	M4	95	M1	140
	M3	180	M3	124	M3	340
	M2	265	M1	71	M2	222

Block III	V1		V2		V3	
	M1	78	M3	196	M2	235
	M3	135	M4	262	M3	260
	M4	130	M1	155	M1	115
	M2	145	M2	220	M4	483

Block IV	V1		V3		V2	
	M1	81	M2	246	M3	296
	M2	175	M3	191	M2	260
	M4	175	M1	145	M1	112
	M3	114	M4	323	M4	450

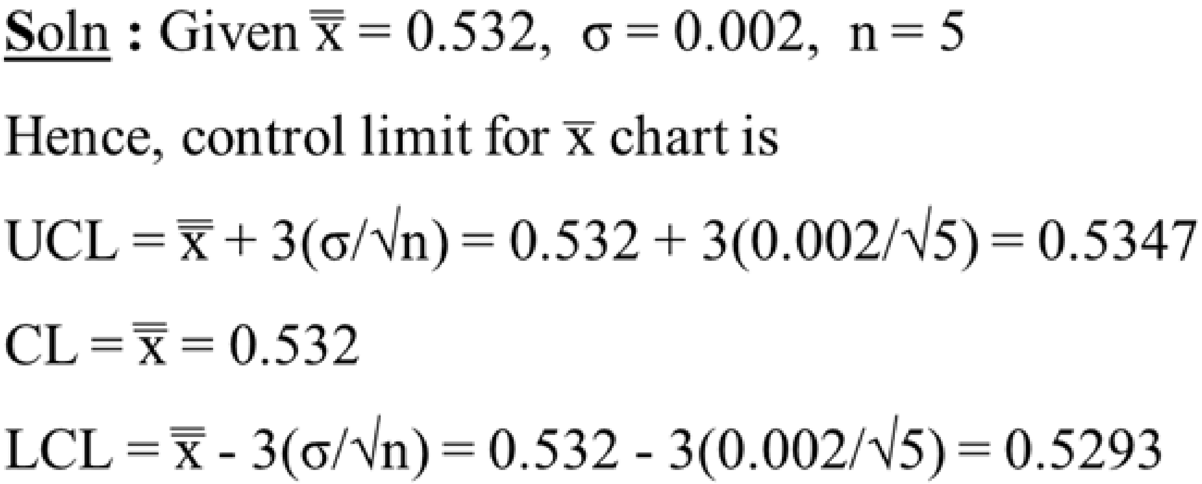
As indicated above **:**
Missing numbers in Block I - V1 : 609 – (94 + 220 + 185) = 110;
Missing numbers in Block II – V1 : 450 – (160 + 124 + 71) = 95;
Missing numbers in Block III – V3 : 1093 – (235 + 260 + 115) = 483;
Missing numbers in Block IV - V1 : = 545 – (175 + 175 + 114) = 81

(ii) Block-variety table

Variety	BLOCK				Total
	I	II	III	IV	
V_1	609	450	488	545	2092
V_2	920	870	833	1118	3741
V_3	1067	1072	1093	905	4137
Total	2596	2392	2414	2568	9970

(iii) CF = $G^2/N = 9970^2/VMR = 9970^2/3x4x4$ = 2,070,852.08333

Ex 13 **:** A machine drills hole in a pipe with a mean diameter of 0.532 cm and a st1andard deviation of 0.002 cm. Calculate the control limits for mean of samples 5.



Soln **:** Given $\bar{\bar{x}} = 0.532,\ \sigma = 0.002,\ n = 5$

Hence, control limit for $\bar{x}$ chart is

$$UCL = \bar{\bar{x}} + 3(\sigma/\sqrt{n}) = 0.532 + 3(0.002/\sqrt{5}) = 0.5347$$

$$CL = \bar{\bar{x}} = 0.532$$

$$LCL = \bar{\bar{x}} - 3(\sigma/\sqrt{n}) = 0.532 - 3(0.002/\sqrt{5}) = 0.5293$$

Ex 14 : The following data gives the readings for 6 samples of size 6 each in the production of a certain product. Find the control limits using mean chart. [Given for sample size n = 6, $A_2 = 0.483$],

Sample	1	2	3	4	5	6
Mean	300	342	351	319	326	333
Range	25	37	20	28	30	22

Soln :

Sample	1	2	3	4	5	6	Total
Mean	300	342	351	319	326	333	1971
Range	25	37	20	28	30	22	162

Hence, control limit for Mean chart is

$\overline{\overline{x}} = (\sum \overline{x})/\text{number of samples} = 1971/6 = 328.5;$

$\overline{R} = (\sum R)/\text{number of samples} = 162/6 = 27$

$UCL = \overline{\overline{x}} + A_2\overline{R} = 328.5 + 0.483(27) = 341.54$

$CL = \overline{\overline{x}} = 328.5$

$LCL = \overline{\overline{x}} - A_2\overline{R} = 328.5 - 0.483(27) = 315.45$

Ex **15 :** The data shows the sample mean and range for 10 samples for size 5 each. Find the control limits for mean chart and range chart and comment on the state of control of the process.

Sample	1	2	3	4	5	6	7	8	9	10
Mean	21	26	23	18	19	15	14	20	16	10
Range	5	6	9	7	4	6	8	9	4	7

Soln :

Sample	1	2	3	4	5	6	7	8	9	10	Total
Mean	21	26	23	18	19	15	14	20	16	10	182
Range	5	6	9	7	4	6	8	9	4	7	65

The control limits for Mean chart ($\bar{x}$-chart) is –

$\bar{\bar{x}} = (\sum x)/\text{number of samples} = 182/10 = 18.2$

$\bar{R} = (\sum R)/\text{number of samples} = 65/10 = 6.5$

$UCL = \bar{\bar{x}} + A_2\bar{R} = 18.2 + 0.577(6.5) = 21.95$

$CL = \bar{\bar{x}} = 18.2$

$LCL = \bar{\bar{x}} - A_2\bar{R} = 18.2 - 0.577(6.5) = 14.45$

The control limits for Range chart is –

$UCL = D_4\bar{R} = 2.114(6.5) = 13.741$

$CL = \bar{R} = 6.5$

$LCL = D_3\bar{R} = 0(6.5) = 0$

Control Limits for Mean chart

$$UCL = \bar{\bar{x}} + A_2\bar{R} = 18.2 + 0.577(6.5) = 21.95$$

$$CL = \bar{\bar{x}} = 18.2$$

$$LCL = \bar{\bar{x}} - A_2\bar{R} = 18.2 - 0.577(6.5) = 14.45$$

Control Limits for R chart

$$UCL = D_4\bar{R} = 2.114(6.5) = 13.741$$

$$CL = \bar{R} = 6.5$$

$$LCL = D_3\bar{R} = 0(6.5) = 0$$

Comparing the Given Sample Data with Control Limits of Mean Chart

Sample	1	● 2	● 3	4	5	6	● 7	8	9	● 10	Total
Mean	21	26	23	18	19	15	14	20	16	10	182
Range	5	6	9	7	4	6	8	9	4	7	65

It is observed that the Sample nos. 2 & 3 are above the UCL and Sample nos. 7 & 10 are below the LCL of Mean Chart. We know that Lack of control in Mean Chart indicates Machine Problems. Hence remedial action should be initiated to rectify the machine problem.

Now, Comparing the Given Sample Data with Control Limits of R Chart

Sample	1	2	3	4	5	6	7	8	9	10	Total
Mean	21	26	23	18	19	15	14	20	16	10	182
Range	5	6	9	7	4	6	8	9	4	7	65

All the sample points are within the control limits of R Chart. We know that Lack of control in Range Chart indicates Workers Problems. Since there is no lack of control in Range Chart, we can infer that there is no problem due to workers.

Ex 16 : The Values of sample mean ($\bar{x}$) and the range (R) for ten samples of size 5 each is tabulated below. Draw mean chart and comment on the state of control of the process.

Sample	1	2	3	4	5	6	7	8	9	10
Mean	43	49	37	44	45	37	51	46	43	47
Range	5	6	5	7	7	4	8	6	4	6

Given the following control chart constraint for : $n = 5$, $A_2 = 0.577$, $D_3 = 0$ and $D_4 = 2.114$.

Soln :

Sample	1	2	3	4	5	6	7	8	9	10	Total
Mean	43	49	37	44	45	37	51	46	43	47	442
Range	5	6	5	7	7	4	8	6	4	6	58

$\bar{\bar{x}} = (\sum \bar{x})$/number of samples = 442/10 = 44.2

$\bar{R} = (\sum R)$/number of samples = 58/10 = 5.8

The control limits for mean chart in next slide –

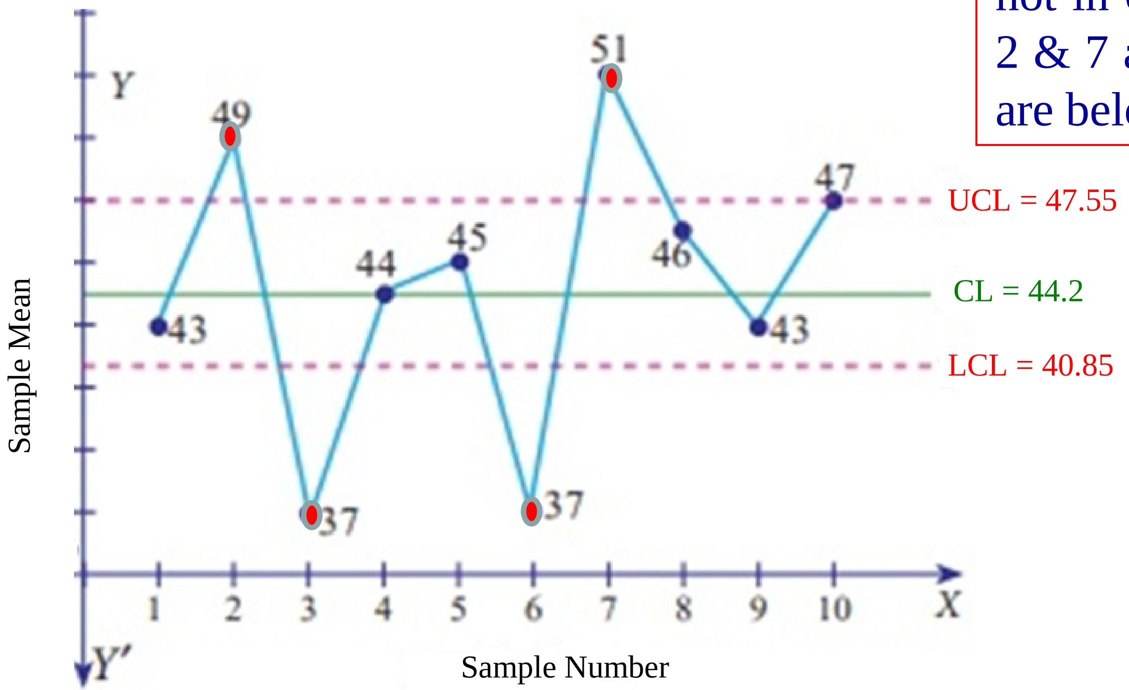
Sample	1	2 🔴	3 🔵	4	5	6 🔵	7 🔴	8	9	10	Total
Mean	43	49 🔴	37 🔵	44	45	37 🔵	51 🔴	46	43	47	442
Range	5	6	5	7	7	4	8	6	4	6	58

$$UCL = \bar{\bar{x}} + A_2\bar{R} = 44.2 + 0.577(5.8) = 47.55$$

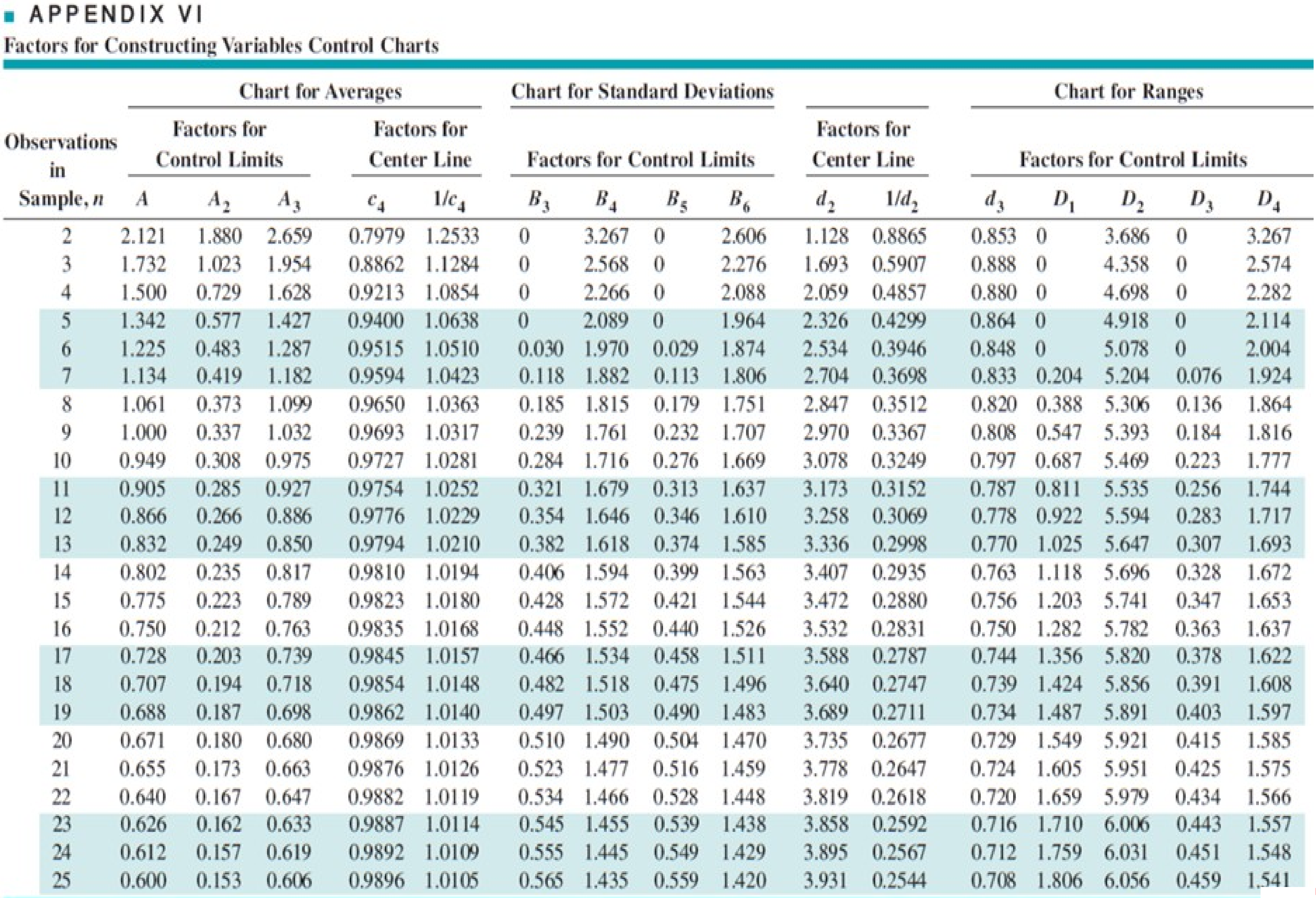
$$CL = \bar{\bar{x}} = 44.2$$

$$LCL = \bar{\bar{x}} - A_2\bar{R} = 44.2 - 0.577(5.8) = 40.85$$

Control Chart



It is evident that the process is not in control - Sample numbers 2 & 7 are above UCL and 3 & 6 are below LCL.



APPENDIX VI

Factors for Constructing Variables Control Charts

	Chart for Averages					Chart for Standard Deviations						Chart for Ranges				
	Factors for Control Limits			Factors for Center Line		Factors for Control Limits				Factors for Center Line		Factors for Control Limits				
Observations in Sample, n	A	A_2	A_3	c_4	$1/c_4$	B_3	B_4	B_5	B_6	d_2	$1/d_2$	d_3	D_1	D_2	D_3	D_4
2	2.121	1.880	2.659	0.7979	1.2533	0	3.267	0	2.606	1.128	0.8865	0.853	0	3.686	0	3.267
3	1.732	1.023	1.954	0.8862	1.1284	0	2.568	0	2.276	1.693	0.5907	0.888	0	4.358	0	2.574
4	1.500	0.729	1.628	0.9213	1.0854	0	2.266	0	2.088	2.059	0.4857	0.880	0	4.698	0	2.282
5	1.342	0.577	1.427	0.9400	1.0638	0	2.089	0	1.964	2.326	0.4299	0.864	0	4.918	0	2.114
6	1.225	0.483	1.287	0.9515	1.0510	0.030	1.970	0.029	1.874	2.534	0.3946	0.848	0	5.078	0	2.004
7	1.134	0.419	1.182	0.9594	1.0423	0.118	1.882	0.113	1.806	2.704	0.3698	0.833	0.204	5.204	0.076	1.924
8	1.061	0.373	1.099	0.9650	1.0363	0.185	1.815	0.179	1.751	2.847	0.3512	0.820	0.388	5.306	0.136	1.864
9	1.000	0.337	1.032	0.9693	1.0317	0.239	1.761	0.232	1.707	2.970	0.3367	0.808	0.547	5.393	0.184	1.816
10	0.949	0.308	0.975	0.9727	1.0281	0.284	1.716	0.276	1.669	3.078	0.3249	0.797	0.687	5.469	0.223	1.777
11	0.905	0.285	0.927	0.9754	1.0252	0.321	1.679	0.313	1.637	3.173	0.3152	0.787	0.811	5.535	0.256	1.744
12	0.866	0.266	0.886	0.9776	1.0229	0.354	1.646	0.346	1.610	3.258	0.3069	0.778	0.922	5.594	0.283	1.717
13	0.832	0.249	0.850	0.9794	1.0210	0.382	1.618	0.374	1.585	3.336	0.2998	0.770	1.025	5.647	0.307	1.693
14	0.802	0.235	0.817	0.9810	1.0194	0.406	1.594	0.399	1.563	3.407	0.2935	0.763	1.118	5.696	0.328	1.672
15	0.775	0.223	0.789	0.9823	1.0180	0.428	1.572	0.421	1.544	3.472	0.2880	0.756	1.203	5.741	0.347	1.653
16	0.750	0.212	0.763	0.9835	1.0168	0.448	1.552	0.440	1.526	3.532	0.2831	0.750	1.282	5.782	0.363	1.637
17	0.728	0.203	0.739	0.9845	1.0157	0.466	1.534	0.458	1.511	3.588	0.2787	0.744	1.356	5.820	0.378	1.622
18	0.707	0.194	0.718	0.9854	1.0148	0.482	1.518	0.475	1.496	3.640	0.2747	0.739	1.424	5.856	0.391	1.608
19	0.688	0.187	0.698	0.9862	1.0140	0.497	1.503	0.490	1.483	3.689	0.2711	0.734	1.487	5.891	0.403	1.597
20	0.671	0.180	0.680	0.9869	1.0133	0.510	1.490	0.504	1.470	3.735	0.2677	0.729	1.549	5.921	0.415	1.585
21	0.655	0.173	0.663	0.9876	1.0126	0.523	1.477	0.516	1.459	3.778	0.2647	0.724	1.605	5.951	0.425	1.575
22	0.640	0.167	0.647	0.9882	1.0119	0.534	1.466	0.528	1.448	3.819	0.2618	0.720	1.659	5.979	0.434	1.566
23	0.626	0.162	0.633	0.9887	1.0114	0.545	1.455	0.539	1.438	3.858	0.2592	0.716	1.710	6.006	0.443	1.557
24	0.612	0.157	0.619	0.9892	1.0109	0.555	1.445	0.549	1.429	3.895	0.2567	0.712	1.759	6.031	0.451	1.548
25	0.600	0.153	0.606	0.9896	1.0105	0.565	1.435	0.559	1.420	3.931	0.2544	0.708	1.806	6.056	0.459	1.541

ALL THE BEST
for
ENSUING COMPREHENSIVE EXAMS

Thank You